DEEPCHECKERS

Deep Learning Based Turkish Checkers Game

by

Cem Güleç - 150117828
Büşra Gökmen - 150116027
Muhammet Şeramet - 150115069

CSE497 / CSE498 Engineering Project report submitted to Faculty of Engineering in partial fulfillment of the requirements for the degree of

BACHELOR OF SCIENCE

Supervised by:

Asst. Prof. Mehmet Kadir Baran

Marmara University, Faculty of Engineering

Computer Engineering Department

2021

DEEPCHECKERS

Deep Learning Based Turkish Checkers Game

by

Cem Güleç - 150117828
Büşra Gökmen - 150116027
Muhammet Şeramet - 150115069

CSE497 / CSE498 Engineering Project report submitted to Faculty of Engineering in partial fulfillment of the requirements for the degree of

BACHELOR OF SCIENCE

Supervised by:

Asst. Prof. Mehmet Kadir Baran

Marmara University, Faculty of Engineering

Computer Engineering Department

2021

ABSTRACT

Deep Learning has become one of the pioneer methods used in many different research fields. What it brought as feature, enabled scientists and engineers to bring different perspectives into problems that were considered as impossible before. On the other hand, considering the recent improvements done at those methodologies, board games that were existing for centuries became one of the increasingly focused subjects among scientists. Reasons for this interest can be listed as, games can be easily implemented for the test environment, it is really hard to come up with a solution that solves it and their popularity over the world is very high.

One of the milestones considered in this field, on the games of chess, achieved by DeepBlue project winning against former world champion Garry Kasparov at a tournament. Later on, due to its complexity research focus has been locked on the game of Go. Another milestone occurred in 2016, where AlphaGo project won against 18-times world champion Lee Sedol. With the outcome from the success at the game of Go, scientists are currently focused on more generalized ways of implementing these methodologies into multi-disciplinary research fields.

In our project, we are focused on a game called “Dama” (Turkish Checkers) which is a game sinking into oblivion. By the help of modern machine learning techniques and combining these techniques with a self-play concept, that is to let game play by itself without any knowledge given, a model has been created and will be improved with best of our efforts.

ACKNOWLEDGEMENTS

We are especially thankful to our advisor Dr. Mehmet Kadir Baran for the infinite amount of resources he has been given to us. Also, we appreciate all of the faculty members of the Department of Computer Engineering for their considerations and advices.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES and FORMULAS

1. INTRODUCTION

1.1 Problem Description and Motivation

Over the last years, deep learning has become one of the pioneer methods to be used in many fields. It brought a new aspect to those fields where researchers slightly gave up, having a thought that there will be no hope of having a satisfactory improvement. Board games like Chess, Go and Checkers provided an efficient testing environment for studying the learning, managing and decision-making aspects of these systems. Among the reasons for choosing these games as a field of study, the fact that these games are popular in the world and that they are played widely has contributed to its selection as a field of study. Another factor is that these games have a set of rules that restrict the players' appropriate behavior (i.e. legal moves), thus simplifying the problem in this case. The third factor is that these strategy games have the goal of winning the game for the players to achieve. Prizes are awarded to players who demonstrate the best and correct game strategies under the constraints of limited resources that allow players to reach the set goal.

Board games such as Chess and Go have become the focus of attention due to the diversity of the moves they have and the complexity of the game. While these games continue to maintain their popularity over time, the game of Checkers has lagged behind the technological developments. Although there is good development in the fields of AI and ML, unfortunately it cannot attract the attention of researchers due to the worldwide popularity of the game. The main aim of this project is to solve the Turkish Checkers game by combining these areas in a sophisticated and modern way.

1.2 Aims of the Project

Main aim of this project is to create an AI model (agent) that can learn and play Turkish Checkers with deep learning-based techniques.

The minor objectives of this project are:

Preventing Turkish Checkers from losing its value by creating an advanced competitor.

- As a result of research that has been made, it is clearly seen that the reason for the decrease in the number of people interested in Turkish checkers is that people do not have a good rival in the game. One of the minor aims in our project is to increase the number of players in Turkish checkers by creating a good opponent that challenges players.

To create a new survey area that focuses on Turkish Checkers.

- While researching studies on Turkish checkers, the number of surveys we came across was almost zero. One of our minor goals is to reawaken people's interest in Turkish Checkers with the innovations we have made in this field and to ensure that it can be used as a source for new researches in this field.

Model (Game Agent) Strategic Learning

- Models (agents) learn board coordinates for Turkish Checkers game and gain the ability to move on the board.
- Models (agents) to learn the strategies and best possible moves of the Turkish Checkers game.

Model Self Game Learning and Optimization

- The models (agents) learn the game by playing with the best model in the previous episode without manually playing with the human. With this approach, it is aimed to learn the game by playing by itself.

Model Predictive Ability

- Models (agents) learn more moves depending on the number of games and decide which of the next moves will be best. In this way, it is aimed to give the models the ability to predict.

2. DEFINITION OF THE PROJECT

2.1 Scope of the Project

2.1.1 Inner Scope

Our main goal in this project is to propose an intellectual method for board games. Using neural networks to extract information from a large body of information in the environment of how to play checkers at expert level without having any dataset one of our goals is to learn about the game in terms of features that are believed to be important. Deep Learning Techniques and MCTS methods were applied to achieve this goal. There are the same problems using Neural Networks [1] and Feed Forward Neural Networks [2], but the solutions they offer, existing technologies. That's why we decided to focus on using Convolution Neural Networks in our project.

2.1.2 Constraints/Limits

AlphaZero achieves great results and definitely outperforms other studies. Lots of computational resources alongside previous approaches required to take advantage of this algorithm.

- Training time depends on the computer's processing power.
- The system should give output in a short time.
- The output of the system should be reasonable.

2.1.3 Claims/Assumptions

- It is assumed that the end model will be accurate. In a given game state, the model will play the most promising move.
- It is assumed that the system will be more efficient compared to previous works.

2.2 Success Factors and Benefits

2.2.1 Measurability/Measurement Success:

- The success rate of this project is based on the results of the games played until the end of the checkers program. It was measured by looking at the loss values of the model.
- The model achieved a reasonable level of gameplay without any expert.
- The model can provide an output in a limited time.

2.2.2 Benefits/Effects:

This project offers a new approach to virtual games. With these approaches, games can be automated and presented to players. Different game styles can be added to the game instead of the existing methods. This type of

projects can also provide an incentive for the need for research and development on strategy games. For example, when AlphaGo is seen successfully defeating the grandmasters, people were both respected and interested. Apart from these, the applicability of deep learning approaches can be demonstrated in studies that require strategy.

2.3 Professional Considerations

2.3.1 Methodological considerations/engineering standards

- In our project we used Git as a control system. Git is a version control system that provides strong coordination and communication skills between team members.
- The most suitable and widely used language for deep learning applications is Python. We can easily find many source codes to help on open source platforms. As a result in this project Python is used as the programming language.
- Google Colaboratory(Colab) is used for model development, GPU, and Ram support.
- Google Drive is used for data transfer and storage.
- Flow chart diagrams are used in order to visualize the progress of the project.

2.3.2 Societal / Ethical Considerations

Some of the progenitors of this project [3] [4] are good examples to predict what ethical implications this project will have. After DeepBlue and AlphaGo had successfully won as a machine versus against a human being, some ethical debates ensued, especially over whether we could really define what machines could do in the future. One of the impacts from this discussion is the establishment of a nonprofit research institute called the 'Future of Life Institute (FLI)' by Max Tegmark, a professor of physics at MIT. This institute works to mitigate the existential risks facing humanity, especially the existential risks from human-level artificial general intelligence (AGI). Not to prevent us from improving our understanding about our brain, self-consciousness and intelligence, but to gain control over it.

Building a machine that excels in a human-versus-game may be a small challenge to put our way, but it will ultimately have a profound impact on our understanding of ethics for our lives.

Figure - 1: Beneficial AI 2017 Conference

While doing this project, taking into account social and ethical values, it was taken into consideration not to violate personal data. A social impact, there may be techniques that people will use in strategy games or in business areas where strategy development is important. Economically, this project can reduce game development costs.

The realization of the project does not contain a biological effect and an element that may pose a danger to nature.

2.3.3 Legal Considerations

Our project does not have a legal consideration. The research we use to deepen our understanding of the concepts described is free to use licenses. All frameworks - programs used during this project are open source. Therefore, we will not need a license to develop the tools.

2.4 Literature Survey

Under this section, several works are described depending on their familiarity with the concepts that we used. Besides studying the game Checkers, we also benefited from different games such as Chess, Go, Shogi, Othello these concepts applied to.

In the work of Murray Campbell, A. Joseph Hoane Jr, Feng-hsiung Hsu [5], authors worked on to create a chess playing system with the sole purpose to defeat the human world champion in a regulation match. The Deep Blue project was the culmination of about 15-years of efforts. System was consisting of a single-chip chess search engine, a massively parallel system with multiple levels of parallelism, a strong

emphasis on search extensions, a complex evaluation function and usage of a game database that consists of Grandmaster gameplays information.

First version of Deep Blue failed to accomplish this purpose, losing against former world champion Garry Kasparov in 1996. After that, with received improvement parameters to be handled they enhanced the first version to form Deep Blue II. Followings are the major enhancements done:

- Firstly, a new significantly enhanced chess chip is designed. The new chess chip had a completely redesigned evaluation function, new feature range going around 6400 to over 8000. Furthermore, pet chip search speed increased to 2-2.5 million positions per second with this system.
- Secondly, more than double the number of chess chips in the system, and using the newer generation of SP computers in order to support the higher processing demands.
- Lastly, developing a set of software tools to aid in debugging and match preparation for evaluation tuning and visualization tools.

After the enhancements were completed and optimized for the system requirements, Deep Blue II won a six-game match against the world champion Garry Kasparov in 1997.
It made history as the first computer to win both a chess game and a chess match against a reigning world champion under regular time controls.

In the study of David Silver, Thomas Hubert [6], the authors are working on a game-playing program that replaces traditional game-playing programs with a DNN, RL and game tree search. In the neural network part, they use convolutional neural networks that takes game state and gives a vector of move probabilities for each action and a value that predicts the outcome of the game from that state. Neural networks are trained by reinforcement learning from self-played games. They use a general-purpose MCTS algorithm in their game search trees. As a result, they compared the model to the best-known models in Chess, Shogi and Go respectively. In chess, AlphaZero first surpassed Stockfish after just 4 hours (300,000 steps); On shogi, AlphaZero first outstripped Elmo after 2 hours (110,000 steps); And in Go, AlphaZero first outperformed AlphaGo Lee after 30 hours (74,000 steps).

3. SYSTEM DESIGN AND SOFTWARE ARCHITECTURE

3.1 System Design

System design consists of Self-Play, Neural Network and Arena. In this section these different parts will be explained detailly.

3.1.1 System Model

In the system model part, components are explained. As shown in Figure 2, in the self play stage the model creates its own data set through self played games. Then with these created data sets, the model starts the learning process in the Neural Network part. After learning is completed, the trained model is put against older version of the neural network. Depending on the results of those games, the model decides whether to update its neural network or not.

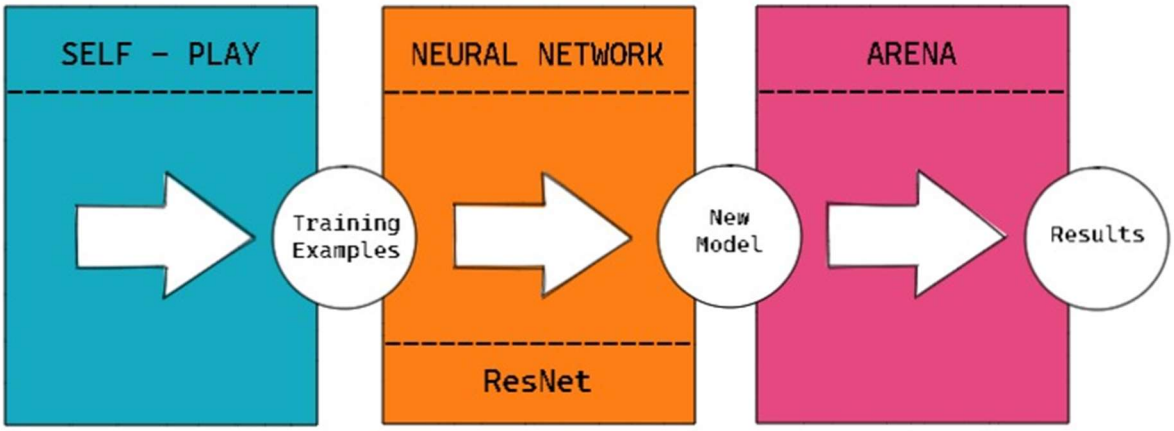

Figure - 2: Representation of the system

Self - Play

At this stage, the model creates data sets by self play starting from tabula rasa (empty state). In these games, the model records each game state, player's turn, all valid actions from that state, and the action that player chosen to play. In order to choose an action from each state, the model uses MCTS with Zobrist and NN. The model continues to play with itself by performing these stages for the second player.

Neural Network

In this step the neural network starts the learning process with the data set obtained from the self - play stage. In order to find the optimal NN structure for the game of Checkers, we have tried lots of variations. The ResNet structured approach gave us the results that we found the most successful among these variations. With the help of ResNet structure, the model was making faster and more visible progress. Even though we chose to work with Google Colab Pro, which gives us very good resources, compared to the related works (section

- 2.4) our resources were incomparably worse. Despite these difficult conditions, we tried to do the best we could with the resources we had and we were able to run the ResNet structure up to 5 iterations in order to see the model progress. In each iteration, the models were playing 100 games in the self - play phase.

Arena

In the arena stage, we put the trained model, with data sets obtained from self - play, in a 40-match game against the untrained version of the model. Depending on the results of these games, if the trained version of the model wins with %60 or more of total games we update our NN with the newer version then we start a new iteration. If the trained model does not successfully pass the specified rate, we do not update the NN structure and go back to the self play stage to start a new iteration.

3.1.2 Flowchart and/or Pseudo Code for Proposed Algorithms

There are 3 main parts of our system and interactions between them is shown in Figure 2. As shown in the figure, our model initially creates a data set by playing with itself in the self-play part. In this part, we also apply MCTS in order to find the best promising moves in game states and Zobrist in order to prevent repetition moves occuring during game play. After this part is completed, the model trains itself with the data set created in the self play part. After learning, the trained version of the model plays against the untrained version of the model in the Arena part. If the trained model wins more than %60 of total games played against the untrained version, the neural network updates and starts again from the self play part in the new iteration. If the trained version fails, then the model starts from the self play part in the new iteration without updating the neural network.

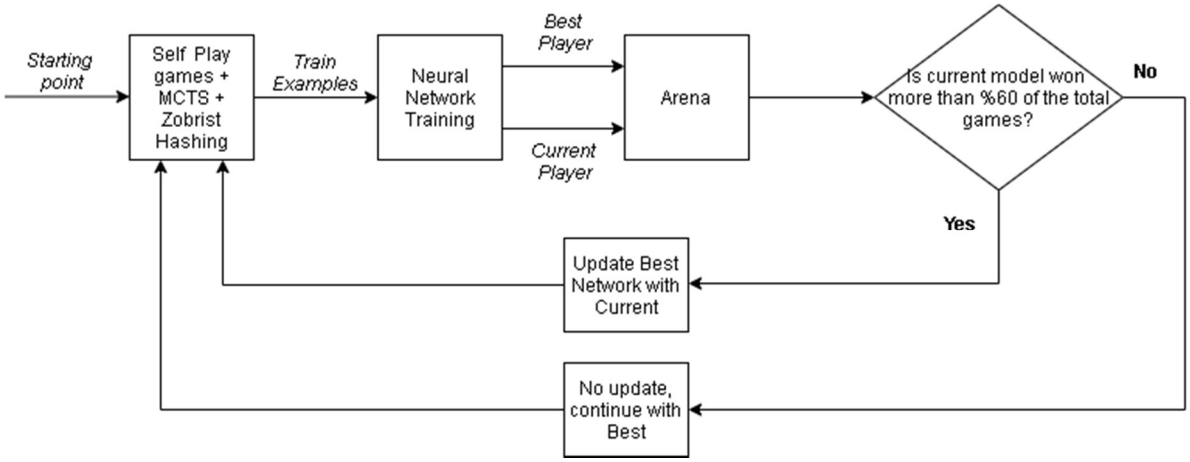

Figure - 3: Flow chart of our model

Algorithm 1 General MCTS approach.

```
function MCTSSEARCH(s_0)
    create root node v_0 with state s_0
    while within computational budget do
        v_l ← TREEPOLICY(v_0)
        Δ ← DEFAULTPOLICY(s(v_l))
        BACKUP(v_l, Δ)
    return a(BESTCHILD(v_0))
```

Figure - 4: Pseudocode of MCTS

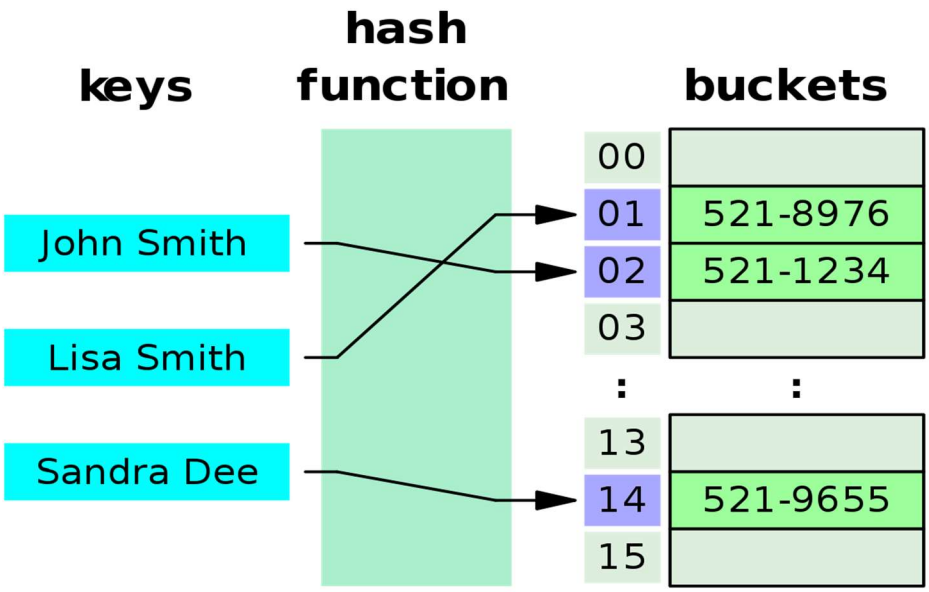

Figure - 5: Flow of Zobrist Hashing

3.1.3 Comparison Metrics

There are some error calculations for the outputs of deep neural networks. In our project we used the softmax activation function for pi values and mean squared error for v values. The Softmax activation function takes the inputs and returns a probability in the range of 0 to 1. On the other hand Mean Squared Error calculates the loss by averaging the squares of the differences between the target values and the predicted values.

After the calculations are complete, the neural networks update the weights using Adam optimizer. Adam uses the square gradients to scale the learning rate and leverages momentum by using the moving average of the gradient instead of the gradient itself.

3.1.4 Data Set or Benchmarks

No external data set used in our project. We get the data which we will use while training our neural network through self-play games. Therefore, as the number of games increases, the data set also increases. In addition to the game states in these games, we are also adding the vertical mirror symmetry of the game state to increase the data set and speed up the development of our model. Vertical mirror symmetry approach is often used in image processing, but can also be applied to symmetric games.

3.2 System Architecture

In each iteration, the model starts with self-played games. After these games are completed, the model starts the learning process. There are two agents involved in this loop, the best-player and the current-player. The best-player contains the best performing neural network obtained so far and is used to generate the self-play memories. The current-player then re-trains its neural network on these memories and is then pitched against the best-player in the Arena part. If the current-player wins with a certain threshold, the neural network inside the best-player is switched with the neural network inside the current-player, and the loop starts again. The neural network gets better at predicting the value of the game state with each iteration and the likely next moves. Also in all those parts, we use the zobrist hashing method to detect and prevent repetition moves occurred during any play. High level flow chart of this project is shown in Figure 6.

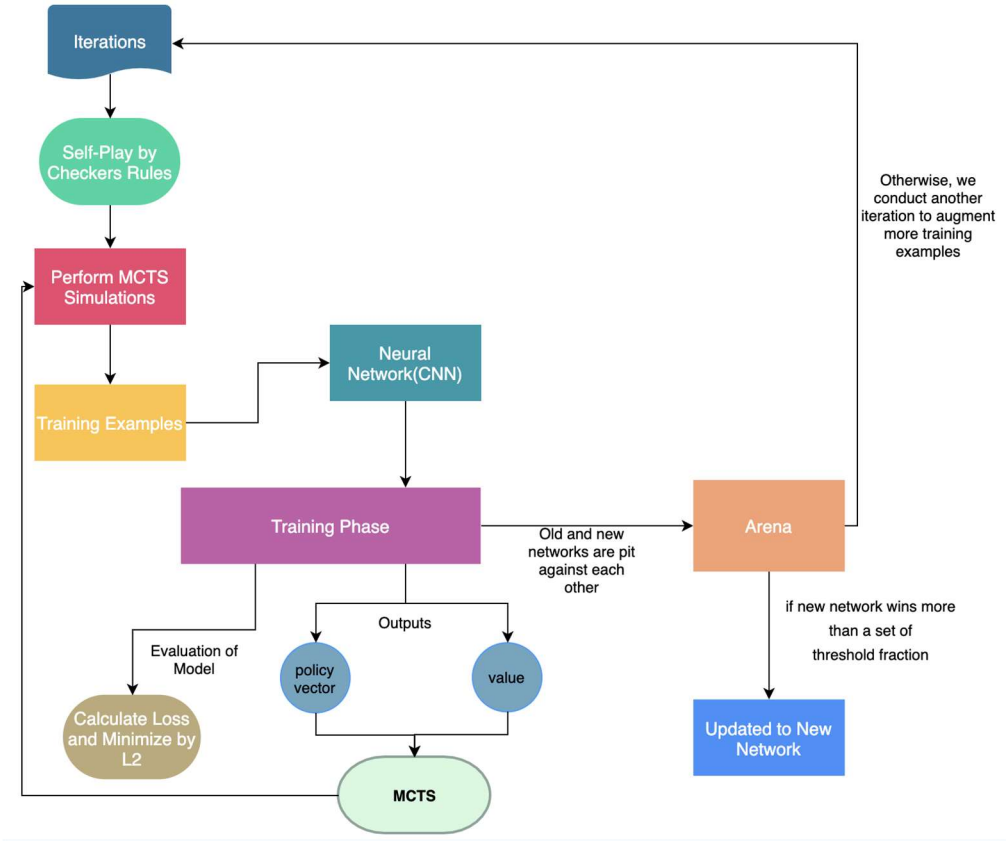

Figure - 6: System Architecture

4. TECHNICAL APPROACH AND IMPLEMENTATION DETAILS

4.1 General Approach

We used the python programming language to implement the algorithms and model. Again, we used Tensorflow and Pytorch frameworks for model implementation. We also used Github to check for code updates and work together more comfortably.

Figure - 7: Icons of python, tensorflow and pytorch

Our modules load the game rules and then iterates through the main loop of the algorithm, which consist of three stages:

1. Self-Play

2. Retraining Neural Network (CNN)

3. Evaluating Neural Network (CNN)

4.2. Neural Network

In our project, we used a convolutional neural network and ResNet approaches in order to find out more suitable NN structure to learn the game better.

Learning from zero, neural network trains with examples obtained from self-play. Input for training of neural networks is a list of training examples, where each example is of the form (board, pi, v). The examples have a board in its canonical form.

- board: current board in its canonical form
- pi: a policy vector for the current board state
- v: a float in range [-1, 1] that indicates which player is closer to win the game

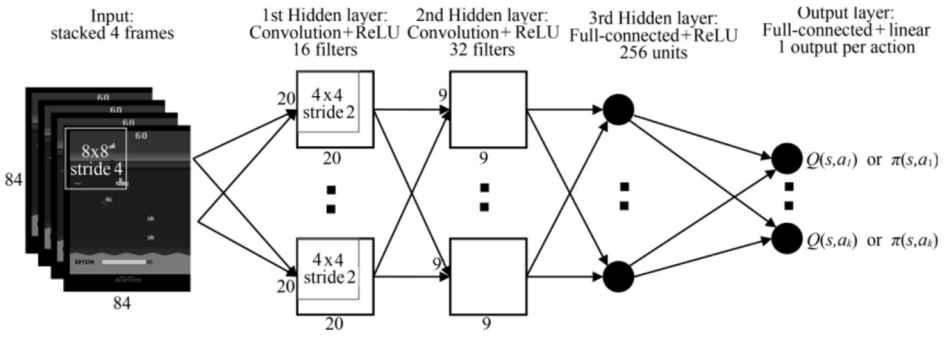

Figure - 8: Convolutional Neural Network structure outlook

Once it has built up enough game positions to fill its memory the neural network will begin training. Through additional self-play and training, it will gradually get better at predicting the game value and next moves from any position, resulting in better decision making and smarter overall play.

4.2.1 CNN General Architecture

A Convolutional neural network (CNN) is a neural network that has one or more convolutional layers and are used mainly for image processing, classification, segmentation and also for other auto correlated data.

We export the input data in the form of two-dimensional matrices to the neural network. The symmetry of the filter to be applied to the two-dimensional information (input data) is taken according to the x and y axis. All values are multiplied element by element in the matrix and the sum of all values is recorded as the corresponding element of the output matrix. This is also called a cross correlation relationship. Let us imagine this computation process as a layer in a neural network. The input data and filter are also actually the matrix of weights updated by continuous backpropagation. The filter, which is a weight matrix for convolution, is shifted over the image in steps of one pixel or larger steps. A scalar b (bias) value is added last to the output matrix applying the activation function. And then, input goes to a fully-connected layer. Fully Connected Layer is simply, feed forward neural networks. Fully Connected Layers form the last few layers in the network. The input to the fully connected layer is the output from the final Convolutional Layer, which is flattened and then fed into the fully connected layer. The output from the final (and any) Convolutional Layer is a 3-dimensional matrix, to flatten that is to unroll all its values into a vector.

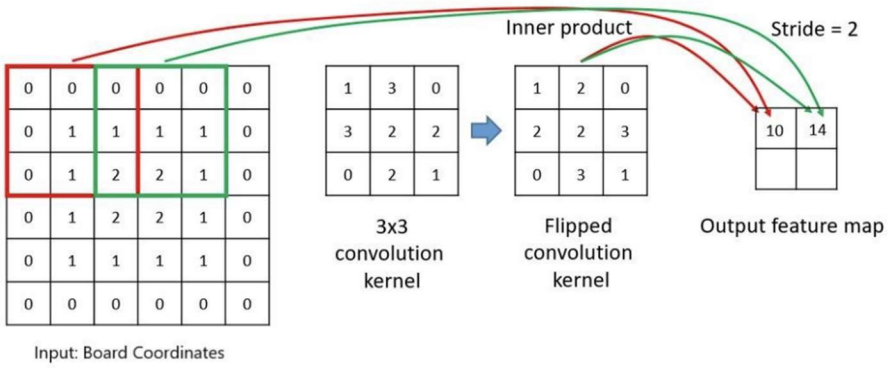

Figure - 9: Convolution process [7]

And finally, a SoftMax layer is added. SoftMax assigns decimal probabilities to each class in a multi-class problem. Those decimal probabilities must add up to 1. This additional constraint helps training converge more quickly than it otherwise would [8].

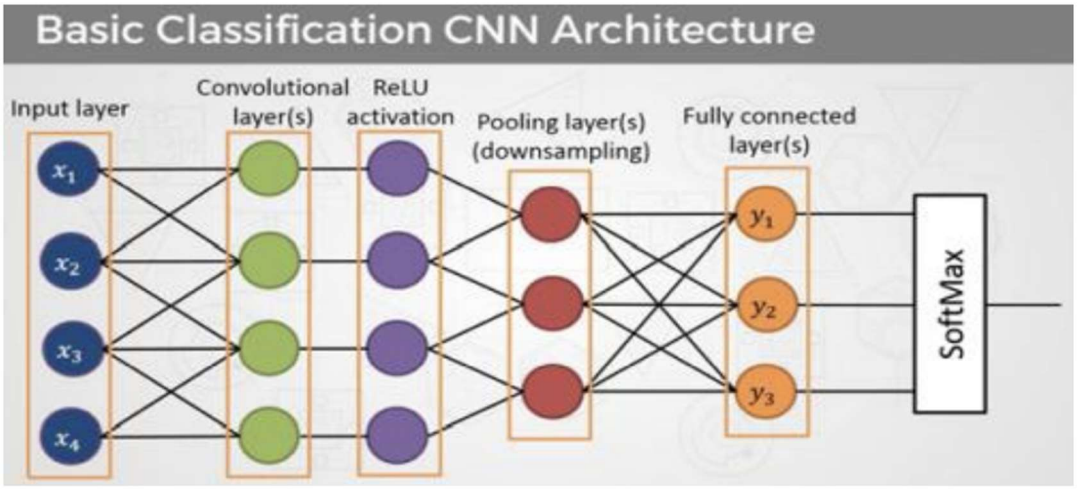

Figure - 10: Classification with CNN Architecture

4.2.2 Residual Networks (ResNet) Architecture

In this architecture, Residual Blocks containing multiple layers are used to reduce the training error. Input x given to the residual block gives a result F(x) after the series of Convolution layer - Batch Normalization layer - Rectified Linear Unit (ReLU) - Convolution layer - Batch Normalization layer. This result is then added to the original x input and expressed as H(x) = F(x) + x. The obtained function is terminated with the rectified linear unit (Figure-11).

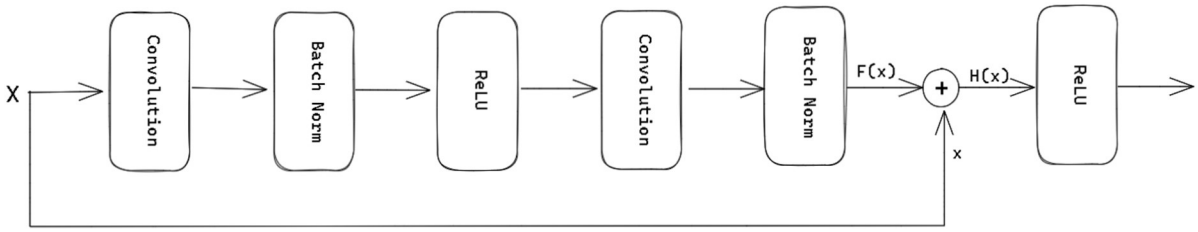

Figure - 11: Residual Block Layers

The ResNet architecture in our project consists of a single Convolution layer and multiple series of Residual blocks. The convolution layer helps us filter the gameplay of the game play structure, while Residual blocks are involved in making the learning process of NN structure faster.

4.3 MCTS

In models based on game playing, an evaluation function is used in the game tree in order to evaluate the game states. Scientists have preferred to use heuristic functions as an evaluation function until recently. With recent studies, a new method has come to the fore. Known as the Monte Carlo style, this method yielded smoother results in terms of systems' evaluation of the game, but fell short of sufficient depth in the game tree. More recently, the Monte Carlo Tree Search was introduced by Rémi Coulom[9]. The popularity of the technique grew rapidly due to its success in the game of Go. It was noteworthy that the technique was not only limited to the game of Go, but also suitable for the general game structure.

Instead of a classic game tree search algorithm, the MCTS algorithm simulates the game multiple times and tries to find the most promising moves based on the simulation results. It is also known as the best first-first algorithm, thanks to its much deeper searches in the game tree.With the evaluation function in continuous development with additional simulations, it will approach the optimum game tree with the given infinite memory and computational power. The integration of the technique is quite simple and consists of 4 steps:

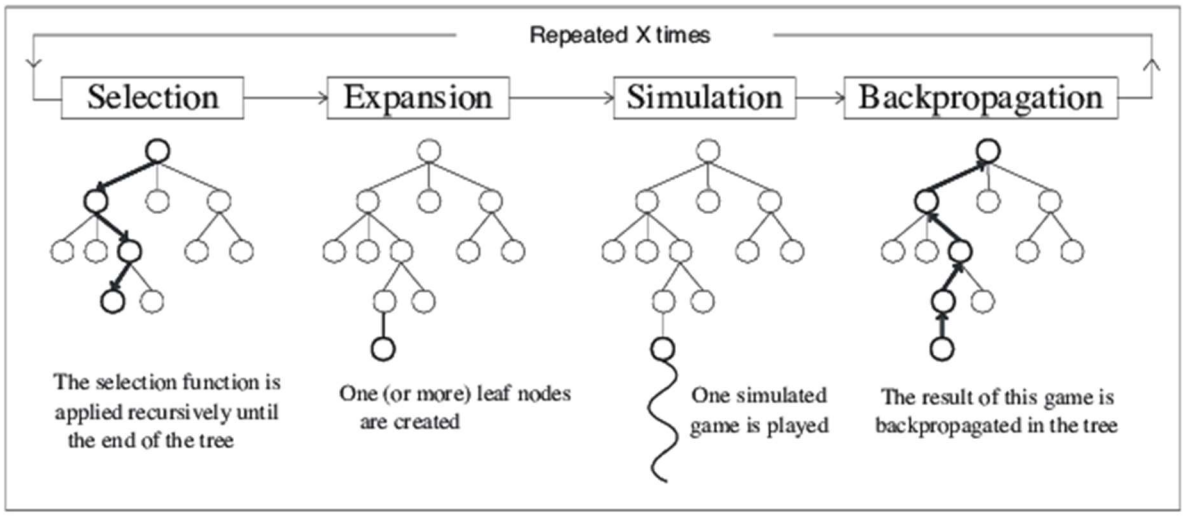

Figure - 12: The schema of the MCTS algorithm

4.3.1 Selection Phase

At this stage, the search starts from the root node (current game state) in the game tree and moves to a leaf node that is not fully expanded. A node that is not fully expanded means that at least one of its children is not expanded.

At each stage of the transition, the node is expanded with the strategy selected for further traversal. The strategy we prefer to use as the selection criterion in our project is the UCT (Upper Confidence Bound) algorithm.This strategy is easy to implement and used in many programs. UCT works as follows:

$$UCT(v_i, v) = \frac{Q(v_i)}{N(v_i)} + c\sqrt{\frac{log(N(v))}{N(v_i)}}$$

Formula - 1: UCT

$Q(v_i)$: corresponds to the number of wins in the child node
$N(v_i)$: is the number of times the child node has been visited
c: is a constant to adjust the amount of exploration
$N(v)$: is the number of times the parent node has been visited

4.3.2 Expansion Phase

In the expansion phase, nodes that are not yet expanded in the game tree are expanded and added to the game tree structure. One or more nodes can be added to the game tree at this stage.

4.3.3 Simulation Phase

In the simulation phase, the information of the node to be added to the game tree is calculated by the Neural Network structure and assigned to the node values.

4.3.4 Backpropagation Phase

At this stage, the node information added in the simulation phase is transferred by updating the values of the other node information on the same path it follows till to the root node.

4.4 Zobrist Hashing

In computer science, a hash is a special data structure that can be used to map a given key to a value. A hash function is used to replace the key with a newly generated hash value.

The Zobrist Hashing algorithm was designed by Albert Lindsey Zobrist in 1969 [10]. It can be widely used in board games to detect repetitions of the board position or to prevent repetitions from occurring. The fact that it is easy to implement in terms of algorithms and the fact that the hash values are almost impossible to overlap with each other makes this algorithm attractive and applicable even in encryption structures used in the field of security.

In our project, the Zobrist Hashing method is used as follows:

- On the 8x8 game board, each square is assigned 4 random values between 1 and 2^{32}. These 4 values represent respectively: white pieces, black pieces, white kings and black kings.
- Since the state of the board will change after any move in the game, a new hash value must be calculated for this. For this, in the previous step, randomly generated numbers are used depending on which squares the stones in that state are in and which stone type they correspond to in those squares. The number obtained as a result of connecting each of these numbers with the OR gate will be the hash value we get for that board state. An example can be seen in Figure 15.
- Since we will get a unique hash value for each board state, it will be possible for us to check whether the board state that our next move will create has been encountered before.
- We designed this algorithm to work integrated with the Monte Carlo tree. While in the Expansion Stage on the Monte Carlo tree, whether a new node will be added or not is controlled depending on whether the Hash value of this board state has been produced before.
- If such a hash value does not exist in the hash table before, the hash value corresponding to the new board state is added to the hash table. Also, the related node structure is added to the Monte Carlo tree. See Figure 15.
- If such a hash value exists before, no such node is added to the tree. This ensures that already computed and already existing nodes in the tree are not recomputed and saves us a computational burden. Because sequential repetitions will increase the depth of the tree considerably and will cause errors such as stack overflow after a while since we do not have unlimited memory. An example can be seen in Figure 16.

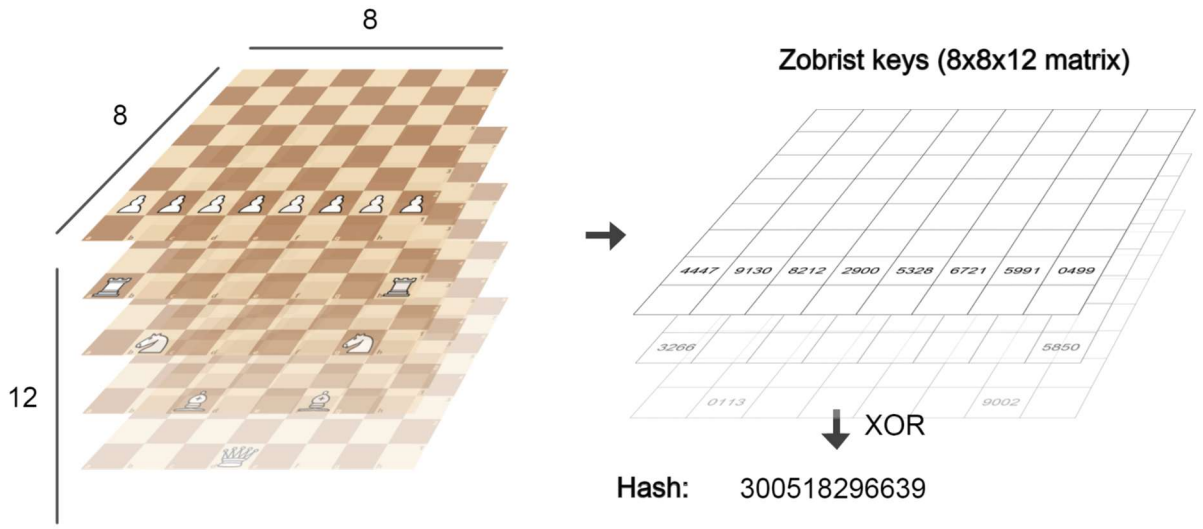

Figure - 13: Application of Zobrist Hashing method on the game of Chess

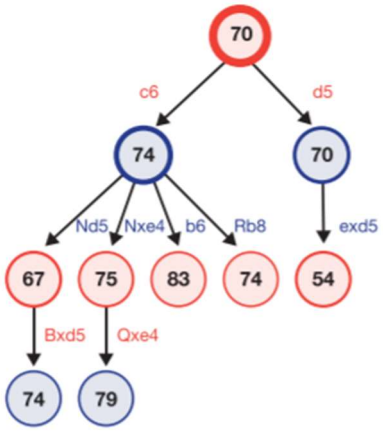

Figure - 14: MCTS without repetition

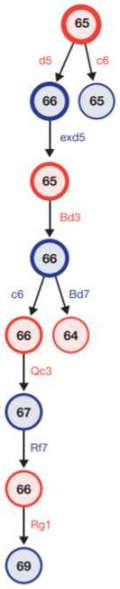

Figure - 15: MCTS and Zobrist with repetition

5. EXPERIMENTAL STUDY

5.1. CNN Experiments

We used two different convolution neural network based model approaches in order to learn the checkers game better.

Input Format:

board: current board in its canonical form

Output Format:

pi: a policy vector for the current board- a numpy array of length

v: a float in [-1,1] that gives the value of the current board

First Experiment:

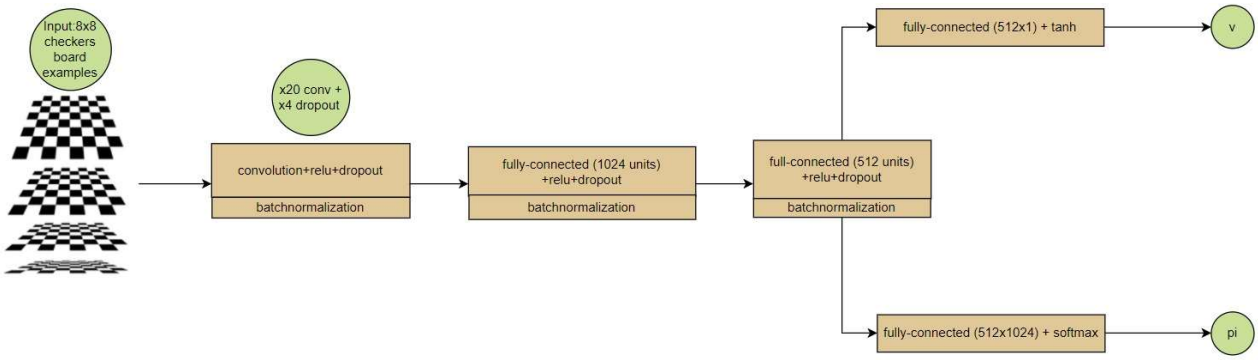

Figure - 16: Architecture used in first experiment

We set up the first neural network structure with 512 filters using four convolutional layers (without dropout).

We set up the neural network structure with 512 filters using 20 convolutional layers and added 4 dropouts (to prevent overfitting) between the convolutional layers.

Parameters for Training Part:

filter numbers: 512
epoch numbers: 10
learning rate: 0.01
batch size: 64
dropout: 0.3

Results:

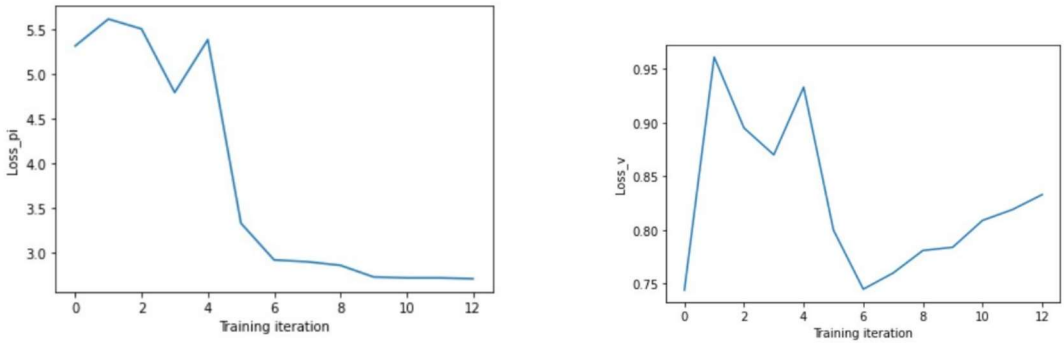

Figure - 17: Pi_loss and v_loss graphs for first experiment

Second Experiment:

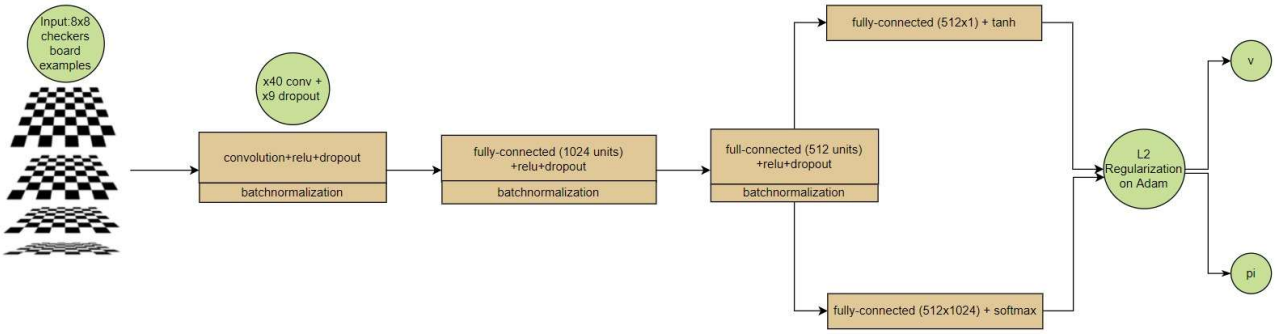

Figure - 18: Architecture used in the second experiment

We increased the total number of convolutional layers to 40. We did this so that the model would increase the depth of the network and generate more invoices. And we added 9 dropouts between the convolution layers (to avoid overfitting). We also added L2 regularization to the optimizer to decrease the loss values.

L2 Regularization

$$\text{Cost} = \underbrace{\sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2}_{\text{Loss function}} + \underbrace{\lambda \sum_{j=0}^{M} W_j^2}_{\text{Regularization Term}}$$

Formula - 2: L2 Regularization

Parameters for Training Part:

filter numbers: 512
epoch numbers: 15
learning rate: 0.001
batch size: 32
dropout: 0.4

Results:

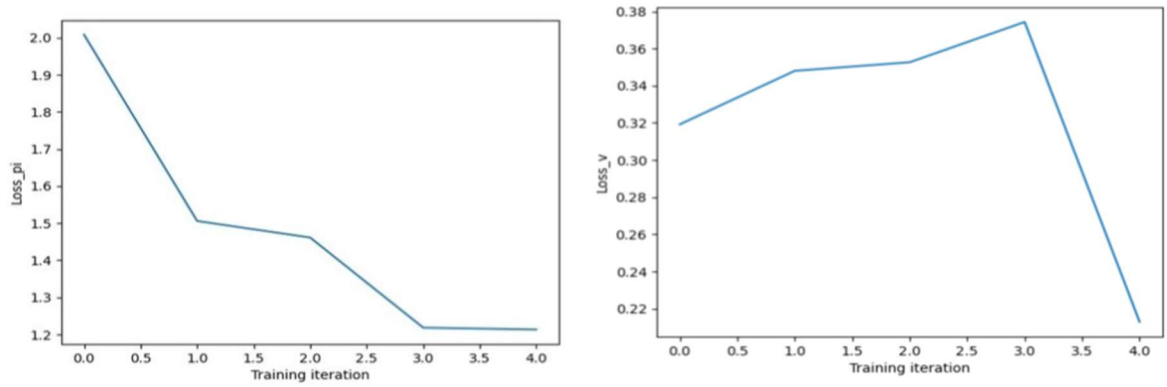

Figure - 19: Pi_loss and v_loss graphs for second experiment

Third Experiment:

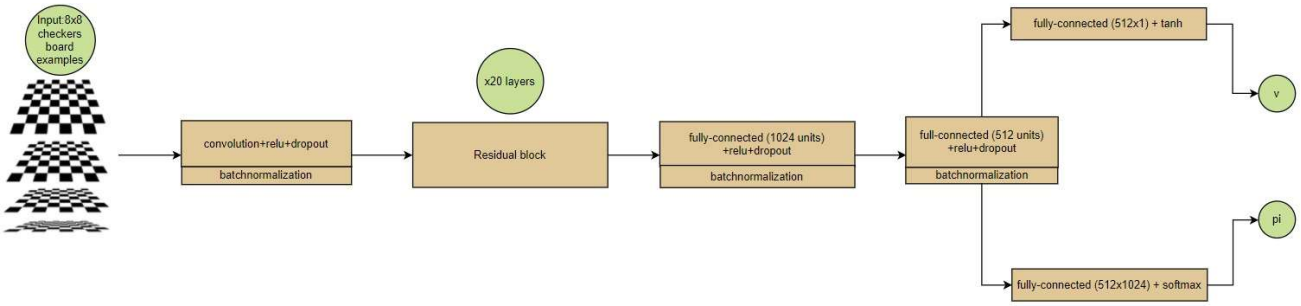

Figure - 20: Architecture used in the third experiment

In the third experiment, since we did not get satisfying results, we have changed our neural network structure by adding 20 ResNet layers into it and changing the number of channels to 128 for the convolutional neural network. At this structure, there are 6.075.535 parameters.

Results:

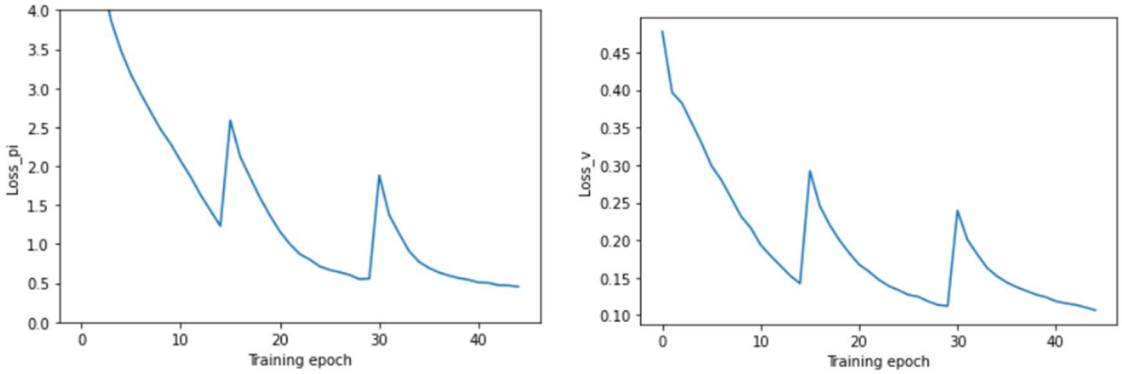

Figure - 21: Pi_loss and v_loss graphs for third experiment

Fourth Experiment:

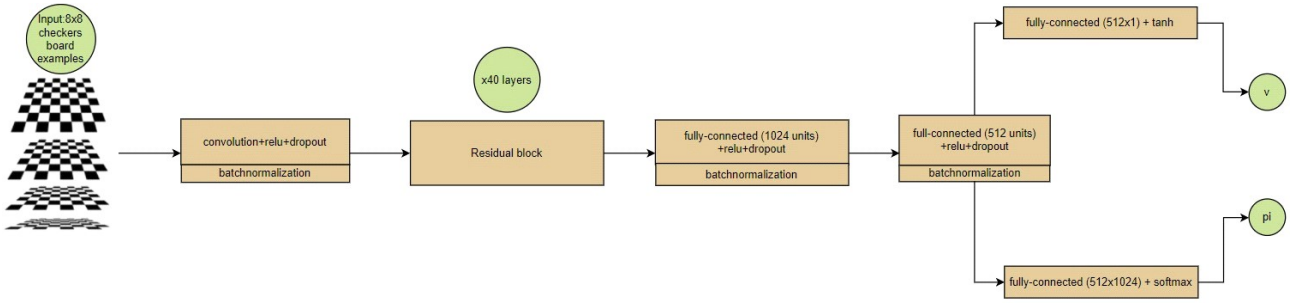

Figure – 22: Architecture used in the fourth experiment

Purpose of the fourth experiment is to test whether there is any significant change in the results by adding more ResNets into our structure and also changing the hyperparameters along with it. For this purpose, the number of ResNet layers increased to 40. At this structure, the number of parameters is 11.999.375.

Results:

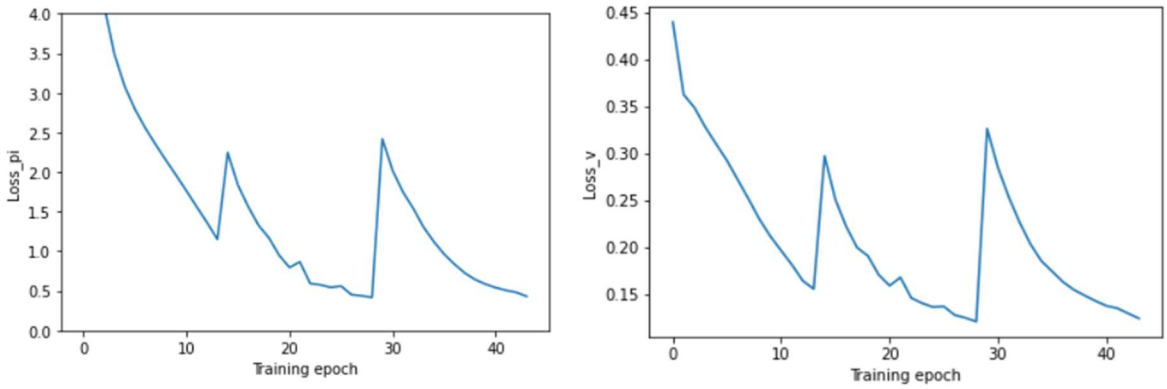

Figure - 23: Pi_loss and v_loss graphs for fourth experiment

As the results show, adding ResNet blocks and also working on hyper-parameters grant us a significant amount of improvement. On the other hand, considering third and the fourth experiments' results, even though they are very close to each other in terms of loss values, there is a significant difference between those models according to their play styles and ability to beat a player in a game.

Therefore, we come up with several conclusions: Firstly, small loss values coming from a small neural network structure does not directly indicate that the model will be performing in a good way. Secondly, even though integrating more ResNet blocks to our structure does not grant us small loss values in the short term, later on it allows us better results in the long run for both loss value and elo rating.

5.2. Mcts and Zobrist

Iteration Number	*Average # of nodes created*	*Average # of repetition*	*Percentage saved*
#1	16806.72	75.90	0.45
#2	49859.86	1859.90	3.73
#3	16915.34	80.57	0.47
#4	17326.12	107.97	0.62
#5	42960.84	793.85	1.84

Table - 1: Savings table acquired by applying zobrist hashing

One of the hardest problems encountered during the project was having repetition moves coming from the nature of game rules of Turkish Checkers. Zobrist Hashing method is used in order to acquire savings in terms of computational power and space. At the above table, for a few iterations, percentages of saving are indicated.

Here, a crucial thing to be specified is that, at a case where the number of repetitions occured and Monte Carlo Tree branching were very high, detecting those repetition moves was very hard. After preventing repetitions from happening, computational and spatial savings have reached around 3.73%.

6. CONCLUSION AND FUTURE WORK

Considering the results displayed at the experiment section, there are a few things worth mentioning.

First and foremost, as we were analyzing related works such as the projects done by DeepMind company, it is a fact that loss value is not the only thing that we should depend on while giving our final thoughts. Alongside that, the more training examples passed to the model for training, better strategies it creates in the long run. Consequently, this process requires long training times, computers with high computational power and a great amount of space for the transfer learning phase. To give an instance to this case, at AlphaZero project in order to achieve professional level performances, DeepMind worked with 4 TPUs, 64 GPUs and 19 CPUs parallely for 40 days of training time.

Our experiments were done in an environment where we used our local computers and parallely worked on Google Colab for testing purposes. As a result, we came up with good results and a model that is performing better than an average player for sure. However, we foresee that, if we were granted with those resources and be able to test our implementation with them, we could easily beat up our current version and would be able to come at a good spot.

REFERENCES

[1] K. Chellapilla, D. B. Fogel, “Evolving an expert checkers playing program without using human expertise”, in IEEE, vol. 5, issue 4, pp. 422-428, Aug 2001.

[2] K. Chellapilla, D. B. Fogel, “Anaconda defeats Hoyle 6-0: a case study competing an evolved checkers program against commercially available software”, in IEEE, July 2000.

[3] Google Developers, “Machine Learning Crash Course - Multi-Class Neural Networks: Softmax”

[4] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ... & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science, 362(6419), 1140-1144.

[5] Murray Campbell, A.Joseph Hoane, Feng-hsiung Hsu, “Deep Blue”, Artificial Intelligence, Volume 134, Issues 1–2, 2002.

[6] David Silver, Thomas Hubert, Julian Schrittwieser, "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm", Science

[7] Ayyuce Kizrak, “Evrişimli Sinir Ağları”, May 2018

[8] Google Developers, “Machine Learning Crash Course - Multi-Class Neural Networks: Softmax”

[9] Coulom, R. (2006, May). Efficient selectivity and backup operators in Monte-Carlo tree search. In International conference on computers and games (pp. 72-83). Springer, Berlin, Heidelberg.

[10] Zobrist, A. L. (1970). Feature extraction and representation for pattern recognition and the game of Go. The University of Wisconsin-Madison.