Universiteit Leiden

Master Computer Science

Curriculum Learning in the game of Go

Name:	Koen Oppenhuis
Student ID:	s1692836
Date:	24/10/2025
Specialisation:	Artificial Intelligence
1st supervisor:	Aske Plaat
2nd supervisor:	Thomas Moerland

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Einsteinweg 55
2333 CC Leiden
The Netherlands

Abstract

AlphaGo Zero learned to play the game Go with deep neural networks only by using reinforcement learning. This was a shocking discovery, but done by using weeks of computing power from the servers of Google Deepmind. To reduce the computing power needed to train such networks, we propose a method using currciculum learning during training. In this method, we start training a network on a small Go board. The knowledge gathered on the small Go board is then transferred to a network training on a bigger Go board. We show 2 different transfer techniques which give networks using our method a head start, compared to networks trained using normal training methods. We come to the conclusion that increasing the size of the Go board during training could have a positive impact on the performance of the network, in particular during the early stages of training.

Contents

1 Introduction

In 2016 Google Deepmind shocked the Go scene by creating a Go playing program that defeated 9-dan professional Lee Sedol. This was the first time a computer program beat one of the best players in the game of Go. The program, called AlphaGo, was a deep neural network trained for multiple weeks on high end GPUs [13]. Although this was a huge accomplishment, it was in part only possible because of the immense computing budget available to the Google Deepmind team. In this thesis, the learning process of a neural network trained with the same techniques as AlphaGo is going to be researched. In particular, we are going to look at using curriculum learning to enhance the speed of the learning process. After learning Go on a smaller Go board, we will use transfer learning to transfer the knowledge to a network learning Go on a bigger Go board. This is similar to how humans learn to play Go. Starting with playing on a 5×5 Go board can introduce you to the rules of the game and the basic tactics. After learning these basic tactics, you can scale up to bigger Go boards, and introduce yourself to more complex strategy. The code used for this thesis can be found on GitHub[1].

1.1 Problem Statement

The question we want to answer in this thesis is as follows:
Does increasing the size of the Go board during training have a positive impact on the performance of a neural network trained with reinforcement learning?
To find an answer to this question, we came up with the following research questions:

Research question 1

What neural network architecture performs best on small Go boards?

Research question 2

Is it possible to transfer knowledge gathered on smaller Go boards to networks training on bigger Go boards?

Research question 3

Which transfer learning method performs best in transferring knowledge from smaller Go boards to networks training on bigger Go boards?

1.2 Methods

To find an answer to these research questions, we will implement multiple reinforcement learning models, inspired by AlphaGo Zero [15], which is a follow

[1] https://github.com/KoenOppenhuis/Curriculum_Learning_in_AlphaGo_Zero

up program of AlphaGo by Google Deepmind. The models will be trained on 5×5 Go boards, 7×7 Go boards and 9×9 Go boards. We will compare the performance of the models that are pre-learned on a small Go board, with the models that started learning directly on a bigger Go board. We will also investigate the difference between two separate methods of transfer learning and see which method has a better performance.

1.3 Upcoming sections

In the following section, the basics of the game Go will be explained, and we will take a look at the complexity of Go on smaller Go boards. We will also explain the basics of reinforcement learning, and clarify the game playing algorithm Monte Carlo Tree Search. In Section 3, we will explain the most important parts about AlphaGo Zero, and go into previous research done on curriculum learning and transfer learning. Section 4 describes the implementation and the experimental design of the research done in this thesis. The following results from these experiments can be seen in Section 5. Finally, in the last Section, we will discuss these results and answer our research questions. We will also note some limitations and ideas about any future work.

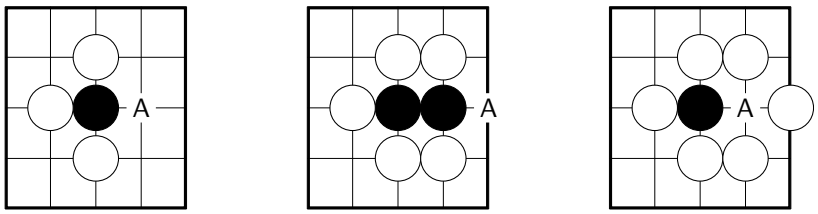
2 Preliminaries

2.1 Rules of the game Go

Go is a game played by two players. The objective of the game is to control more territory than your opponent. You do this by surrounding an area of the board with your stones. You and your opponent alternate in placing a stone on an empty intersection of the board. At the end of the game, the winner is determined by who controls the most space on the board.
The size of a Go board for professional events is 19×19. However, this is not mandatory. Go can be played on any size board, and for this thesis we will be focusing on boards up to 9×9 due to computational reasons. In Section 2.2 we will go into more detail about the complexity of the different size Go boards.

2.1.1 Capturing

In Go, it is possible to capture the stones of your opponent. You can capture one stone, or an entire group of stones. You do this by placing your own stones on the open intersections next to the opponents group. These intersections are called *liberties*. If a group has no liberties left, then that group will be captured, and all stones of that group will be taken of the board. If a group is threatened to be captured, when it has only one liberty left, this group is in *atari*. It is not allowed to place a stone on an intersection that would self capture your group. So you cannot place a stone on an intersection that gives your group 0 liberties. In Figure 1, you can see two examples of groups in *atari*, and an example where it would be illegal to place a stone on intersection A.



(a) (b) (c)

Figure 1: In example (a) and example (b) the black stones are in *atari*. In these examples, if white places a stone on intersection A, they would capture all black stones. In example (c) it is illegal for black to place a stone on intersection A. If white places a stone on intersection A, they will again capture the black stone.

2.1.2 The *ko*-rule

Because of the capturing of stones, positions could be repeated. If this would be allowed, games could go on forever. To counteract this, the *ko*-rule exists.

This rule does not allow the repetition of positions. A position from where repetition is possible is called *ko*. In Figure 2, you can see an example of a basic *ko*-position. Black can capture the white stone by playing on intersection B. If black does this, his stone on B is now in *atari*. However, white cannot recapture this stone, because then the position would be repeated.

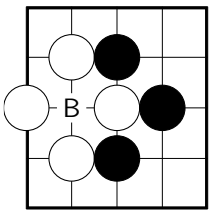


Figure 2: An example of a basic *ko*-position. Black is allowed to capture the white stone, by placing a stone on intersection B. White can't recapture the black stone on intersection B, because then the position would be repeated.

2.1.3 *Komi*

The black player has an advantage over the white player, because black is the first player to place a stone. To make the game more equal, the white player gets a few points for free. This is called *komi*. A fair value for *komi* is different per board size. In order to make it impossible to draw, *komi* consist of a number of full-points combined with a half-point, for example 7.5.

2.1.4 End of the game

If a player doesn't see any helpful moves, they can choose to pass their turn. If both players pass consecutively, the game ends and the counting of score begins. There are a few different scoring methods in Go, of which two methods are mostly used. These scoring methods are Japanese scoring and Chinese scoring. In Japanese scoring, you get points for the territory you control and for the number of stones you captured from your opponent. In Chinese scoring, you get points for the territory you control, but instead of the captured stones from you opponent, you get points for the number of stones you have on the board. In general these scoring methods give the same outcome of a game, but it might sometimes differ by a point. In this thesis we will make use of the Chinese scoring method.

2.2 Complexity of the game Go on smaller board sizes

As said in the previous Section, we will not use the standard Go board size of 19×19. We don't have the computing power available to train neural networks for this board size due to its complexity. The number of legal board positions

Game	Board size	number of legal positions	Branching factor
Go	3×3	1.27×10^4	-
Go	5×5	4.14×10^{11}	-
Connect Four	7×6	4.53×10^{12}	4
Go	7×7	8.37×10^{22}	-
Go	9×9	1.04×10^{38}	-
Chess	8×8	4.59×10^{44}	35
Go	13×13	3.72×10^{79}	-
Go	19×19	2.08×10^{170}	250

Table 1: Estimation of the complexity of differently sized Go boards in comparison to other games [1][17]. Go on a 5×5 Go board has a lower number of legal positions than Connect Four. However, its branching factor is probably higher. The same can be said about Go on a 9×9 Go board, in comparison to chess.

on the 19×19 is around 2.08×10^{170} [17]. Sizing the board down to 9×9 reduces the number of legal positions enormously, to around 1.04×10^{38}. Not only the number of legal positions is much smaller, but also the branching factor is lower. The average branching factor of a game is the average number of available moves per position. The average branching factor for Go on a 19×19 board is around 250 [1]. There is not a lot of research for smaller Go boards, but the maximum number of possible moves on a 9×9 Go board is 81, at the beginning of the game. After each move this number gets lower, so the average branching factor for Go on a 9×9 board also lies lower. In Table 1, the number of legal positions for different sized board is compared to the complexity of other games. If the branching factor is known, then this is also mentioned.
Go on a 7×7 and a 9×9 board is still to complex to solve. However, the 5×5 Go board is solved [18]. In 2003, van der Werf et al. showed that with perfect play, black gets 25 points more than white. This is the entire board, so with perfect play, all white stones can be captured.

2.3 Reinforcement learning

In reinforcement learning, an agent aims to learn the best actions to get the maximum reward for all states in a certain environment. The agent does this by trying out actions and gathering their rewards. The agent doesn't need to have any prior knowledge over the environment. It learns by trial and error.

2.3.1 Markov Decision Process

A reinforcement learning problem can often be modeled as a Markov Decision Process. In a Markov Decision Process the next state depends on the current state, and the actions that can be taken from this state.
A Markov Decision Process for reinforcement learning is formally defined as a

5-tuple $\{\mathcal{S}, \mathcal{A}_s, \mathcal{T}_a, \mathcal{R}_a, \gamma\}$ [11]. Below is explained what each symbol means, with a short example how this could be applied to Go.

- $\mathcal{S}$ is the set of all states in an environment. In Go this would refer to every possible board configuration. For the 5×5 Go board, the number of board configurations is around 4×10^{11}.(See Table 1)
- $\mathcal{A}_s$ is the set of all actions from a certain state s. In Go, this would be the set of empty intersections, where it is legal to place a stone of your color.
- $\mathcal{T}_a(s, s')$ is the transition function. It describes the probability of getting from state s to state s', given action a. Go isn't stochastic, and therefore, this probability is always either 1 or 0.
- $\mathcal{R}_a(s, s')$ is the reward given after the transition form state s to state s'. In Go, this value is only known at the end of the game. If the game is won, the reward will be 1 and if the game is lost, the reward will be -1.
- $\gamma \in [0, 1)$, is a discount factor. This is used in problems that do not have a clear ending. With a γ close to 0 only the immediate rewards are used, while with a γ close to 1 also all previous rewards are taken into account. The use of γ is not needed for Go, because a game of Go has a clear ending. Therefore, γ is not taken into account in this thesis.

2.3.2 Policy function

In reinforcement learning, an agent chooses its actions based on its policy function π. A policy function maps the set of all states to a probability distribution over the set of all actions:

$$\pi : \mathcal{S} \rightarrow p(\mathcal{A})$$

In this thesis the policy function is a neural network. The input for this neural network is the current position on the Go board, the state. The output of the network is a probability distribution of all possible moves in the current position, the actions.

2.3.3 Value function

To evaluate certain states an agent uses a value function $\mathcal{V}$. A value function maps the set of all states to a real number:

$$\mathcal{V} : S \rightarrow \mathbb{R}$$

Just as for the policy function, the value function is also a neural network with as input the current position on the Go board. The output of this network is a value in between -1 and 1. -1 is a prediction that the game is completely lost, and 1 predicts that the game is going to be won.

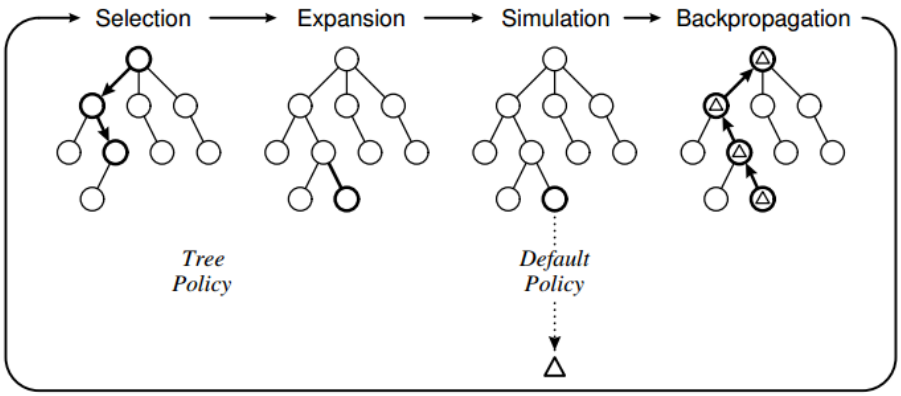


Figure 3: The Monte Carlo Tree Search loop. First a node is selected based on its Tree Policy. This node is expanded with a random unexplored child move. From this new node, the game is played out randomly. The result of this random playout is backpropagated from the expanded node to the root.

2.4 Monte Carlo Tree Search

For my bachelor thesis, I used different Monte Carlo Tree Search algorithms to create an agent for playing the game of ColorShapeLinks [10]. The following explanation is taken from that thesis.
Monte Carlo Tree Search (MCTS) is a game playing algorithm that uses Monte Carlo simulations to build a partial game tree. In this tree every node represents a game state and every link to a child node represents a move in that game state. Every iteration of the algorithm, the partial game tree grows with one game state. Each iteration exists out of 4 phases.

1. Selection
   In the selection phase, the algorithm selects a node. This node is chosen recursively, based on the Tree policy. The most popular variant of MCTS [4] is Upper Confidence bound for Trees (UCT) [8]. This algorithm uses the UCB1 [2] Tree policy:

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}} \tag{1}$$

   In this equation w_i stands for the number of wins recorded in that node, n_i stands for the number of visits recorded in that node, N_i stands for the number of visits recorded in the parent node, and c is a constant that controls the amount of exploration.
   When a node is selected that still has unexplored moves, the algorithm goes on into the next phase.

2. Expansion

In the expansion phase, a random unexplored child move of the selected node is picked. With this move, a new child node is created.

3. Simulation
   From this child node, a Monte Carlo simulation is played. At the end of this simulation the score is recorded.

4. Backpropagation
   This score is than backpropagated through the tree, updating all win counts and all visit counts in all nodes, that were used to get to the expanded node.

The algorithm ends after a chosen amount of iterations or time. At the end of the algorithm, the child node with the highest visit count is chosen as the move to play.

3 Related work

3.1 AlphaGo Zero

In the beginning of 2016 Google Deepmind published a paper about a computer program using deep neural networks to play the game of Go. This program was called AlphaGo [13]. In march 2016 AlphaGo became the first Go computer program to defeat a 9-dan professional in the game of Go, Lee Sedol. In 2017, Google Deepmind published two follow-up papers in which they describe AlphaGo Zero [15] and AlphaZero [14]. In AlphaGo, the neural network was pretrained with supervised learning, on games played by Go professionals. Then only after the supervised learning, reinforcement learning was used. In AlphaGo Zero the supervised learning part is skipped. AlphaGo Zero is only trained by using reinforcement learning. In AlphaZero this idea was generalized and used onto other games, like chess and shogi. For this thesis, the same ideas as for AlphaGo Zero were used.

3.1.1 Network architecture

In AlphaGo Zero, a neural network is used as policy function and as value function. The base of this network is shared for both functions. At the end of the network, there are two different heads, each with a different output. The policy head outputs move probabilities, the policy, and the value head outputs a value, the evaluation of the current board position. The input of this network is not just the current position on a Go board. To account for the *ko*-rule, the previous 8 positions are taken as input.
The input is processed by a convolutional block existing of a convolutional layer, with batch normalization and a rectifier nonlinearity layer. This is followed by the core of the network, which exists out of 39 residual blocks. These residual blocks have 2 convolutional layers, each with batch normalization and rectifier nonlinearities. The convolutional layers in the convolutional block and the residual blocks all use 256 filters with a kernel size of 3×3 and a stride of 1.
After the residual blocks, the network splits to the two different heads. The policy head uses a small convolutional layer with 2 filters of kernel size 1×1 and a stride of 1, then a batch normalization and rectifier nonlinearity layer, and eventually a fully connected layer that has an output size of $19^2 + 1$. This corresponds to every potential move on the 19×19 Go board.
The value head uses a small convolutional layer with 1 filter of kernel size 1×1 and a stride of 1, followed by a batch normalization and rectifier nonlinearity layer. This is followed by a fully connected layer to a hidden layer with size 256, with a rectifier nonlinearity layer thereafter. This is followed with a fully connected layer to a scalar, which is followed by a tanh nonlinearity layer. This last layer makes it so the output falls in the range $[-1, 1]$.

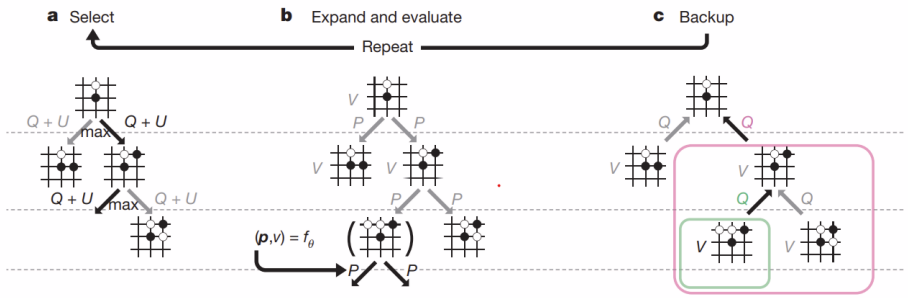


Figure 4: Monte Carlo Tree Search as used in AlphaGo Zero [15]. In **a**, a node is selected by maximizing $Q + U$. Q is the mean value in that node based on previous iterations. U is a value based on the policy network and the visit count of the node. Then in **b**, the tree is expanded, and the new position is evaluated by the value network. Lastly, in **c**, this value V is backpropagated through the tree, and all Q values are updated.

3.1.2 Self-play

The network gets trained by playing games against itself. During these self-play games, the network chooses its moves by means of Monte Carlo Tree Search (MCTS)(see Figure 4). During the simulation phase, instead of a Monte Carlo simulation, the value network is used to evaluate a certain position. This value is backpropagated through the tree.
The policy network is used during the selection phase of MCTS to guide the algorithm to the strongest moves. AlphaGo Zero uses a variant of the PUCT algorithm [12]. During the selection phase an action is selected at timestep t, by maximizing the following equation:

$$a_t = \underset{a}{\operatorname{argmax}}(Q(s_t, a) + U(s_t, a)) \tag{2}$$

In this equation, $Q(s_t, a)$ is the mean value of action a in node s_t. $Q(s_t, a)$ is comparable with the first part of Equation 1, where the value of a node is equal to the win rate of that node. $U(s_t, a)$ is described by the following equation:

$$U(s_t, a) = c_{puct} P(s_t, a) \frac{\sqrt{\sum_b N(s_t, b)}}{1 + N(s_t, a)} \tag{3}$$

Here, $N(s_t, a)$ is the number of times action a was visited in previous iterations, $\sum_b N(s_t, b)$ is the number of times all actions from node s_t were visited in previous iterations, and c_{puct} is a constant that determines the amount of exploration. $P(s_t, a)$ is the probability of selecting action a in node s_t. This probability value is calculated by the policy network, and in this way the policy network can steer which moves get explored. $U(s_t, a)$ is comparable with the second part of Equation 1, where the exploration of different moves is regulated.

After a set number of iterations an action is chosen. In MCTS, this is usually the most visited action. This is also the case in AlphaGo Zero, except at the beginning of the game. During the first 30 moves of a game, the so called warm up steps, the action is chosen in proportion to its visit count. This adds diversity to the self-play games while still having a high chance of picking good moves.

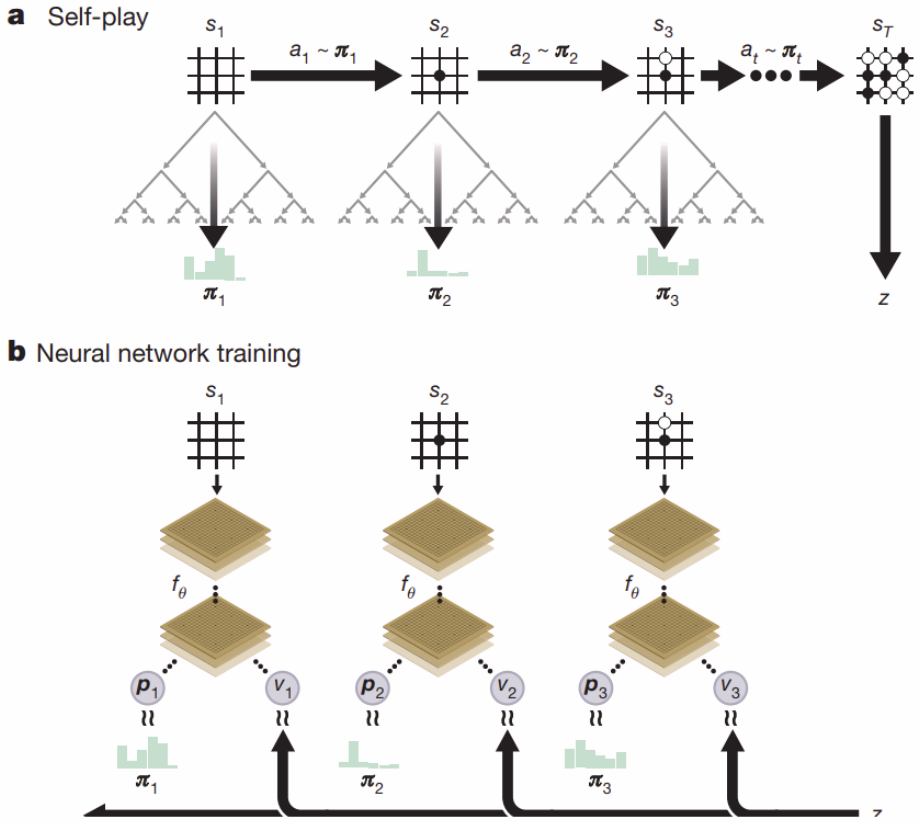


Figure 5: Self-play Reinforcement learning in AlphaGo Zero [15]. **a** During self-play, the move probabilities π and game outcome z are collected. **b** Then, these probabilities and the game outcome are used in combination with the predicted move probabilities and the predicted value to update the neural network parameters using loss function l described in Equation 4.

3.1.3 Training

The self-play games are used to train the policy and value network. Each position during self-play is stored as (s_t, π_t, z_t). s_t is the state during timestep t, π_t is a vector of the move probabilities in state s_t, created by MCTS, and z_t is either -1 or 1 based on the eventual winner of the game.

The neural network is updated to maximize the similarity between neural network move probabilities $\mathbf{p}$ and the MCTS move probabilities π, and to minimize the error between the neural network value v and the self-play outcome z. This is done by adjusting the network parameters θ with gradient descend using the following loss function l:

$$l = (z - v)^2 - \pi^T \ln \mathbf{p} + c||\theta||^2 \tag{4}$$

$(z - v)^2$ is the value loss, $-\pi^T \log \mathbf{p}$ is the policy loss and c is a parameter to control L2 weight regularization. This helps to prevent overfitting. In Figure 5, you can see the self-play and training process.

3.1.4 Evaluation

Before the use of an updated network for generating self-play games, it first needs to prove that it is stronger than the previous network. The updated network plays 400 evaluation games against the previous network. If the updated network wins less than 55% of these games, it's discarded and the previous network is still used for generating self-play games.

3.2 Curriculum learning

With curriculum learning, you try to guide a machine learning algorithm in learning a certain problem. This is done by dividing the task up into smaller sub problems, a curriculum. The idea behind this is that these smaller sub problems are easier to learn, and with the knowledge learned in these sub problems, the normal problem should be easier to solve. Bengio et al. [3] showed promising results on 3 classification problems in which curriculum learning not only has a faster convergence speed, but also has a better performance on the more difficult problems.

3.2.1 Curriculum learning in reinforcement learning

Curriculum learning is also applicable in reinforcement learning. In Narvekar et al. [9], they split up the curriculum learning methodology into three components: Task generation, Sequencing and Transfer learning.

- **Task generation:** Task generation is the process of creating subtasks in which to obtain experience. These subtasks can be created manually or they can be generated dynamically during training. The performance of the curriculum learning is dependent on the quality of these subtasks.
- **Sequencing:** The process of ordering the subtasks created by the task generation is called sequencing. This can be done manually or automatically during training. If subtasks are created to add a bit of complexity to the previous subtask, then an obvious sequencing would be to order these

tasks from least complex to most complex. When subtasks have no overlap with each other, it becomes more difficult to find the best sequencing.

- **Transfer learning:** The subtasks created by the task generation can have a different state space, action space, transition function or reward function. The method of transferring the knowledge gathered on these different subtasks to the final task is called transfer learning.

In AlphaGo Zero curriculum learning is used by means of self-play. Each game played could be seen as a new subtask which is used to train the neural network. Then, for sequencing, they take random samples from the last 500.000 self-play games played. Because these subtasks are using the same state space, action space, transition function and reward function, transfer learning is not needed.

3.3 Transfer learning

Transfer learning is a machine learning technique where a model trained on one task is adapted for a different but related task. There are many areas in which transfer learning is successfully applied, for instance in image classification [7] and in language models [5]. The idea in both image classification and in language models, is to use a machine learning model that already has trained on a general dataset. This model is then transferred and further trained with a different, often more specific, dataset to specialize on that subject.
The models used for image classification are convolutional neural networks. The model used in AlphaGo Zero is also a convolutional neural network. Because transfer learning was successful for the convolutional neural networks used in image classification, we believe this could also be applied to the convolution neural network in AlphaGo Zero.

3.4 AlphaGo Zero version by Michael Hu

For the implementation of the AlphaGo Zero algorithms, we used code written by Michael Hu [6] posted on github. This implementation was made for the 9×9 Go board. Because of the smaller board size compared to AlphaGo Zero, this implementation used smaller networks. After the convolutional block, 10 residual blocks were used instead of 39 and the number of filters used in each convolutional layer was 128 instead of 256. To save computational budget, the evaluation step was also taken out of the algorithm.
After 40 hours of training on a single server equipped with 128 CPUs and 8 RTX 3090 GPUs, this network got the level of an amateur 1 dan player on the 9×9 Go board.

4 Methodology

In this section we are going to explain the structure of the networks and we are going to explain the experiments that were conducted to find an answer to our research questions. For the implementation of the experiments, we adapted code uploaded to github [6] to accommodate for each experiment. The experiments were run on the the National Supercomputer Snellius [16]. The training of each neural network was run for 24 hours on 2 AMD EPYC 9334 processors with each 32 CPU cores and on 4 NVIDIA H100 GPUs. Because of budget limitations, each training run could only be done once.

	AlphaGo Zero	5×5	7×7	9×9
Go board size	19×19	5×5	7×7	9×9
komi	7.5	1.5	7.5	7.5
Residual blocks	39	{5,7,9}	{7,9}	9
Filters	256	64	64	64
Self play games per checkpoint	25,000	5,000	5,000	5,000
MCTS iterations	1600	50	100	200
Warm up steps	30	8	12	16
Uses evaluation	✓	-	-	-

Table 2: Comparison between the parameters of AlphaGo Zero and our experiments. In this thesis, we use smaller networks, with less residual blocks and less filters per layer. We also use fewer MCTS iterations and less warm up steps. This is possible, because of the fact that our experiments are run on smaller Go boards.

4.1 Experimental Design

We are going to run experiments on 3 different Go board sizes, a 5×5 Go board, a 7×7 Go board and a 9×9 Go board. In Table 2, you can see the difference between the most important parameters for these experiments, also compared with AlphaGo Zero.

4.1.1 Baseline experiments

For later experiments, we need to compare all networks to a baseline. That's why we start with training the baseline networks. These networks are trained with no prior knowledge of the game Go, exactly like in AlphaGo Zero.
For the 5×5 Go board, we train 3 different neural networks. 1 network with 5 residual blocks, 1 with 7 residual blocks and 1 with 9 residual blocks. For the 7×7 Go board we train a 7 residual blocks network and a 9 residual blocks network. For the 9×9 Go board, we only train a 9 residual blocks network.

4.1.2 Network transfers

The baseline networks are also used for transfer learning from the smaller boards to the bigger boards. After 3 hours of training a checkpoint of the network is made. This checkpoint is then used as initialization of a new network, that starts training on a bigger board. Because of the difference in state and action space, these networks need to be altered before they can be used on a bigger board. To adapt a network to a larger board size, we use 2 different methods:

1. **Copy method:** This first method copies all weights in the baseline network, that can be used for the network that trains for a larger board size. Because of the convolutional layers, this is almost every layer in the network. The filters in a convolutional layer, do not care about the size of the board that they are used on. This means that the weights in these filters can be copied from one network to another without any dimensional issues. The only layers where the number of weights between different board sizes differs, are in the policy head and the value head. These values will be initialized randomly.

2. **Expansion method:** The second method expands the baseline network with 2 extra residual blocks. First, this method follows the same steps as the copy method. Hereafter, 2 residual blocks are placed in between the last residual block and the policy and value head. These new residual blocks will be initialized randomly.

With these methods, we are going to train 6 neural networks, 3 for the 7×7 Go board and 3 for the 9×9 Go board. In Table 3, you can see exactly which neural networks are going to be trained, and which baseline is used for their initialization.

board size	residual blocks	transfer method	transferred from
7	7	copy	5×5, 7 residual blocks
7	7	expansion	5×5, 5 residual blocks
7	9	copy	5×5, 9 residual blocks
9	9	copy	5×5, 9 residual blocks
9	9	expansion	5×5, 7 residual blocks
9	9	expansion	7×7, 7 residual blocks

Table 3: The networks which are going to be initialized with baseline networks. For the 7×7 Go board, all networks get initialized from baseline networks trained on the 5×5 Go board. For the 9×9 Go board, there is also a network initialized from a baseline network trained the 7×7 Go board.

4.1.3 Tournament ranking

To compare the performance of the different neural networks to each other, tournaments are being played. In a tournament, every participating neural

network plays every other participating neural network for 40 games. 20 games with black, and 20 games with white. At the end of the tournament the win percentage of all games of a participating neural network is used to evaluate the performance of this neural network.
The games in this tournament are played while using warm up steps for the entirety of the game. This means that the neural network has the highest chance to play its top move, but there is a probability that an other move is chosen. If the warm up steps were not used, every match between 2 neural networks would end in the same result.

5 Results

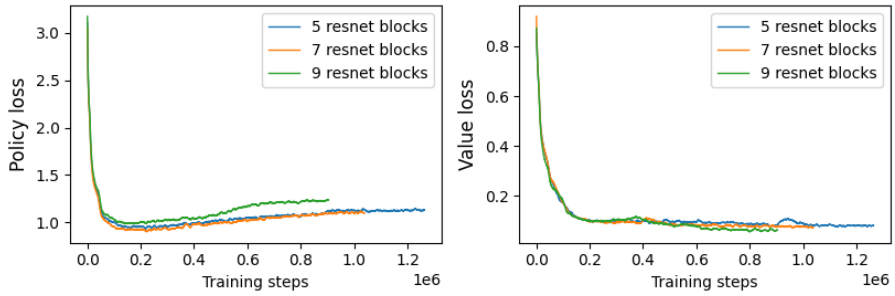
The results will be divided into 3 sections, each section corresponding to a research question asked in Section 1.

5.1 5×5 and 7×7 neural network performance

On the 5×5 Go board, 3 different sizes of neural networks are implemented. The first neural networks has 5 residual blocks, the second has 7 residual blocks and the final network has 9 residual blocks. On the 7×7 Go board, we look at 2 different sizes of neural networks, one with 7 residual blocks, and one with 9 residual blocks.

5.1.1 5×5 Go board

In Figure 6, the loss curves for the networks trained on a 5×5 Go board are shown. In Figure 6a you can see the policy loss of each network, and in Figure 6b, you can see the value loss of each network. The 5 residual blocks network has the most training steps and the 9 residual blocks network has the fewest training steps. This is because of the network size. The smaller networks can play more games against itself because it takes less time to calculate a policy and evaluation. They also need less time to update their weights.
The value loss is a good approximation for the strength of a network. The lower this loss is, the better the network predicts if a game state is winning or losing. Looking at Figure 6b, the 3 networks converge to a value of 0.1 after around 200,000 training steps. The reason this happens quite early in the training process is because of the simplicity of the game on a 5×5 Go board. If black plays a decent strategy, it will almost always win over white.



(a) Policy loss

(b) Value loss

Figure 6: Loss curves of neural networks trained on a 5×5 Go board. The policy loss curves (a) and the value loss curves (b), are almost similar for all networks. The smaller networks can train for more training steps, because they take less time in calculating their policy and evaluation.

To illustrate this, we also ran a tournament, in which 4 checkpoints from every

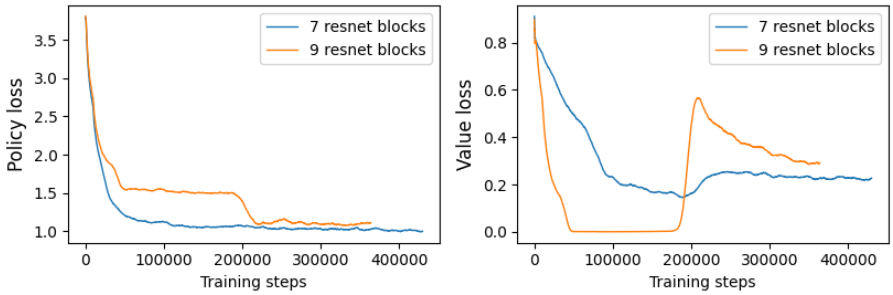
network take part. The checkpoints are taken after 1 hour, 3 hours, 12 hours and 24 hours. These networks play 40 games against every other network, for a total of 440 games. In Table 4, you can see the win percentage of every network in this tournament. Because of the enormous advantage black has in this game, after only one hour of training, the networks are already close to the level of the longer trained networks. It is also interesting to see a slight drop in win percentage in the 24 hours trained networks. This could possibly be explained by overfitting on what they see as the best strategy, which makes the network less robust against different strategy's.

	1 hour	3 hours	12 hours	24 hours
5 residual blocks	47,27%	52,05%	50,23%	50,23%
7 residual blocks	44,77%	52,50%	51,26%	50,91%
9 residual blocks	48,41%	51,26%	52,05%	48,86%

Table 4: Win percentage of neural network checkpoints trained on a 5×5 Go board after playing 40 games against every other neural network checkpoint. Every neural network checkpoint has a win percentage close to 50%. This is caused by the large advantage black has on the 5×5 Go board.

5.1.2 7×7 Go board

In Figure 7, you can see the loss curves for both the neural networks trained on the 7×7 Go board. In Figure 7a you can see the policy loss of each network, and in Figure 7b, you can see the value loss of each network.
The loss curves of the 9 residual blocks network look different in comparison to the 7 residual blocks network and to the loss curves of the networks that trained on the 5×5 Go board. First, let's have a look at its value loss. In a learning network, a downwards trend would be expected in the value loss curve, as can be observed for the 7 residual blocks network. However, for the 9 residual blocks network the loss value drops to 0. After around half the training steps the value-loss shoots up and then starts gradually decreasing. A value loss of 0 means that the network is certain in its prediction about who will be the winner. What happened in this experiment was, that during self-play, the algorithm would pass as white, because it predicted a win because of the *komi*. However after black made some moves, it would realize this tactic was not good and the game would be lost. Instead of finding other moves than passing, the algorithm got into a negative feedback loop and started resigning as white whenever possible. Because a few percentage of the games are non resignable games, the algorithm got out of this loop, and started re-learning on the 7×7 Go-board. In Figure 7a, you can see that this is also the moment that the policy loss drops to the same level as in the 7 residual blocks network.
When we run a tournament with 4 different checkpoints from each network, it is clear that the performance of the 9 residual blocks network is far worse than that of the 7 residual blocks network. In Table 5, you can see all the win



(a) Policy loss

(b) Value loss

Figure 7: Loss curves of neural networks trained on a 7×7 Go board. The value loss curve (b) of the 9 residual blocks network drops quickly to 0. The network evaluated white as always losing and got stuck in a negative feedback loop. It was always resigning as white. After winning a non-resignable game around the half-way mark, the network started re-learning on the 7×7 Go board. The 7 residual blocks network seems to have normal loss curves for both the policy loss (a) and the value loss.

percentages. Contrary to the 5×5 Go board, there can be a correlation observed between longer training time and performance. The win percentages of the 7 residual blocks networks are significantly higher than the win percentages of the 9 residual block network, due to its poor start.

	1 hour	3 hours	12 hours	24 hours
7 residual blocks	38,21%	59,64%	85,36%	98,57%
9 residual blocks	7,86%	9,29%	32,86%	68,21%

Table 5: Win percentages of neural network checkpoints trained on a 7×7 Go board after playing 40 games against every other neural network checkpoint. The 7 residual blocks network outperforms the 9 residual blocks network on every checkpoint. The 7 residual blocks network is improving on every checkpoint compared to their previous checkpoints.

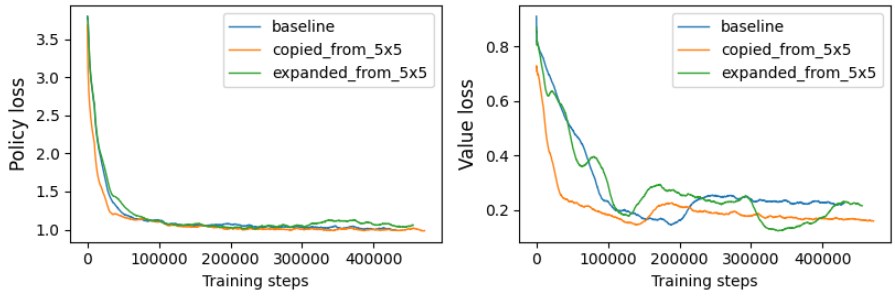
5.2 Knowledge transfer

To see if there is any knowledge transferred from smaller Go boards to larger Go boards, we are going to compare the loss curves of the baseline networks, with the loss curves of the networks that were pretrained on smaller Go boards. We are also looking at the performance of the baseline networks compared to the the networks that were pretrained on smaller Go boards after just a few hours of training. If there is any knowledge transferred to the pretrained networks, then we will see an improvement in performance in the beginning of training over the baseline networks. First we are going to look at the networks trained

on the 7×7 Go board, and thereafter we are going to look at the networks trained on the 9×9 Go board.

5.2.1 7×7 Go board

For the 7×7 Go board, 3 networks with 7 residual blocks are trained, and 2 networks with 9 residual blocks. The 7 residual blocks networks are a baseline network, a network which is copied from a trained network on the 5×5 Go board, and a network that is expanded from a trained network on the 5×5 Go board. The 9 residual blocks networks are a baseline network and a network which is copied from a trained network on the 5×5 Go board.
In Figure 8, you can see the loss curves of the 7 residual blocks networks. In both the policy loss and the value loss, the copied network has lower values at the start of learning. The value loss is almost in its entirety lower, which could indicate a better performance. The policy loss converges with the baseline policy loss. This just indicates how well the network can predict the policy, so it is expected that this converges. The small head start could indicate the recognition of structures it has seen on the 5×5 Go board and choosing to play moves that work well with these structures.
The expanded network does not seem to have a head start in learning compared to the other networks. Also, the value loss curve of this network is fluctuating a bit. It is difficult to say just based on these curves if the expanded network did use any knowledge from the neural network trained on a 5×5 Go board.

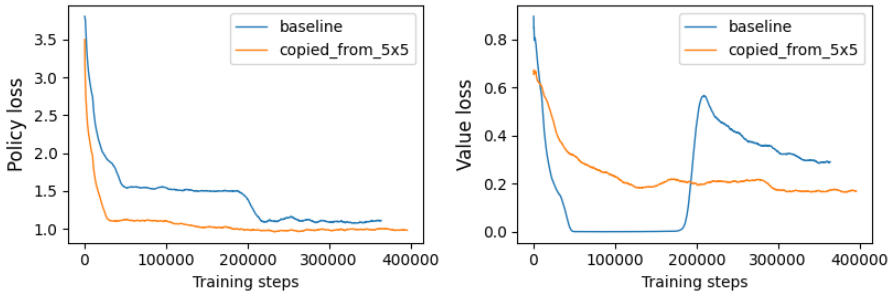


(a) Policy loss

(b) Value loss

Figure 8: Comparison of the loss curves of different 7 residual blocks neural networks on a 7×7 Go board. The policy loss curve (a) of the copied network lies a bit below the other policy loss curves. Also, its value loss curve (b) lies below the other value loss curves. This could indicate that the copied network has transferred knowledge from the network trained on a 5×5 Go board.

In Figure 9, you can see the loss curves of the 9 residual blocks networks. The copied version of the 9 residual blocks has a loss curve similar to the 7 residual blocks networks. So, contrary to the baseline, this network does learn the environment. This could be because of the experience it has from the network trained on a 5×5 Go board.



(a) Policy loss (b) Value loss

Figure 9: Comparison of the loss curves of different 9 residual blocks neural networks on a 7×7 Go board. The loss curves of the copied network follow a normal path, in contrast with the baseline loss curves. There are not enough runs performed to put any conclusions to this observation.

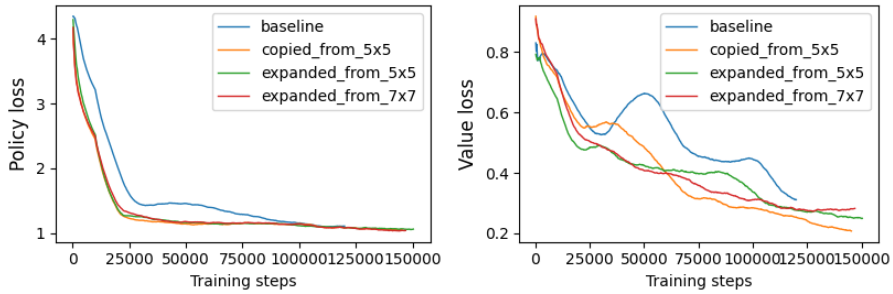
To see if there actually is knowledge transferred to the copied and expanded networks, a tournament was ran on only the 1 hour and 3 hour checkpoints of these networks and the baseline network. In Table 6, you can see that both the copied networks have a higher win percentage after 1 hour, than the baseline and expanded networks. The copied networks don't have any randomly initialized residual blocks, and can therefore directly use the knowledge from the 5×5 Go board onto the 7×7 Go board. This shows that there is knowledge being transferred from the networks trained on the 5×5 Go board. After 3 hours, the expanded network catches up a bit to the copied networks, and also has a better win percentage than the baseline network. However, because of the small difference in win percentage, we can't say for sure that there is also knowledge being transferred in the expanded network.

	1 hour	3 hours
7-blocks baseline	11,07%	48,93%
7-blocks copied	**57,50%**	**86,43%**
7-blocks expanded	5,36%	60,71%
9-blocks copied	50,71%	79,29%

Table 6: Win percentages of neural network checkpoints after 1-3 hours of training time on a 7×7 Go board after playing 40 games against every other checkpoint. The copied network checkpoints have a higher win percentage than the baseline network checkpoints. This indicates that the copied network used the knowledge it had from the network trained on a 5×5 Go board.

5.2.2 9×9 Go board

For the 9×9 Go board, 4 networks with 9 residual blocks are trained. 1 of these networks is the baseline network, 1 network is copied from the 9 residual blocks network trained on the 5×5 Go board, 1 network is expanded from the 7 residual blocks network trained on the 5×5 Go board and 1 network is expanded from the 7 residual blocks network trained on the 7×7 Go board. There is no copied version from the 9 residual blocks network trained on the 7×7 Go board, because of its failed start.
In Figure 10, you can see the loss curves of these networks. In Figure 10a, the policy loss curves of the copied and expanded networks go down a lot steeper in the beginning, than the baseline loss curve. In Figure 10b, the value loss curves of the copied and expanded networks also descend quicker than the baseline loss curve. Just as in the experiments on the 7×7 Go board, this indicates that there is information transferred from the smaller Go boards to the larger Go board. The difference with the experiments on the 7×7 Go board is that the expanded networks seem to behave similarly to the copied network. A possible explanation for this difference could be in the percentage of the network that is randomly initialized. For the expanded network trained on the 7×7 Go board, $\frac{2}{7}$ of the network was randomly initialized. For the expanded networks trained on the 9×9 Go board, only $\frac{2}{9}$ of the network was randomly initialized.



(a) Policy loss

(b) Value loss

Figure 10: Comparison of the loss curves of different 9 residual blocks neural networks on a 9×9 Go board. The policy loss curves (a) of the copied and expanded networks go down steeper at the beginning of training. This indicates that these networks use the knowledge transferred from the networks trained on smaller Go boards. The value loss curves (b) of the copied and expanded networks are lower in general. This could also be an indication of better evaluation of the 9×9 Go board because of the knowledge transferred from the networks trained on smaller Go boards

To see if there is transfer learning going on, again a tournament was ran on only the 1 hour and 3 hour checkpoints of these networks. The win percentages of the networks can be seen in Table 7. The win percentages of the copied and expanded networks are higher than that of the baseline network. This indicates

that there is knowledge being transferred from the smaller Go boards to the 9×9 Go board. In contrast with the 7×7 experiments, the expanded networks, now have a similar or better performance than the copied network.

	1 hour	3 hours
baseline	10,00%	16,07%
copied from 5×5	33,93%	81,79%
expanded from 5×5	**55,00%**	**87,86%**
expanded from 7×7	34,29%	81,07%

Table 7: Win percentages of neural network checkpoints after 1-3 hours of training time on a 9×9 Go board after playing 40 games against every other checkpoint. The copied and expanded network checkpoints outperform the baseline checkpoints. Even after 1 hour of training the copied and expanded networks have a higher win percentage than the baseline network after 2 hours of training. This indicates that the copied and expanded networks use the knowledge transferred from the networks trained on the smaller Go boards.

5.3 Performance evaluation

To evaluate the performances of the networks that are pretrained on smaller Go boards, we are going to look at their playing strength after 12 hours and after 24 hours. First we are going to look at the networks trained on the 7×7 Go board, and thereafter we are going to look at the networks trained on the 9×9 Go board.

5.3.1 7×7 Go board

In Table 8, you can see the win percentages of the 12 hours and 24 hours checkpoints of the networks trained on the 7×7 Go board when playing against each other. For the 12 hours checkpoints, the copied and expanded networks have a higher win percentage than the baseline network. However, after 24 hours of training this is flipped. The baseline network outperforms the copied and expanded networks. A possible explanation for this, could be that the pretrained networks get stuck in local minima. Because they play suboptimal moves with knowledge from the small boards, instead of exploring other potential strategies. It could also be possible that networks with these sizes are reaching the limit for this environment.

5.3.2 9×9 Go board

In Table 9, you can see the win percentages of the 12 hours and 24 hours checkpoints of the networks trained on the 9×9 Go board when playing against each other. For the 9×9 Go board, after 12 hours of training, the baseline performs worse than the copied and expanded networks. The head start from training on the small Go boards still gives a big advantage after 12 hours. After

	12 hours	24 hours
7-blocks baseline	25,36%	**83,21%**
7-blocks copied	**39,29%**	54,29%
7-blocks expanded	35,00%	67,86%
9-blocks copied	33,93%	61,07%

Table 8: Win percentages of neural network checkpoints after 12-24 hours of training time on a 7×7 Go board after playing 40 games against every other checkpoint. After 12 hours of training the copied and expanded networks have a higher win percentage. This is probably still due to the head start with the transferred knowledge from the networks trained on the smaller Go boards. After 24 hours, the baseline network has a higher win percentage. This could be due to the other networks getting stuck in local minima.

24 hours this advantage is much smaller, and compared to the expanded version, the performance of the baseline is similar. The copied network is a bit better than the other networks. Looking at Figure 10b, the value loss curves of the networks did not yet converge. Therefore it is still a possibility that any of the networks can get a better performance after converging.

	12 hours	24 hours
baseline	5,71%	68,93%
copied from 5×5	**39,64%**	**76,79%**
expanded from 5×5	37,50%	66,79%
expanded from 7×7	37,50%	67,14%

Table 9: Win percentages of neural network checkpoints after 12-24 hours of training time on a 9×9 Go board after playing 40 games against every other checkpoint. After 12 hours, the copied and expanded networks outperform the baseline network, indicating that the knowledge transferred from the smaller Go boards is still beneficial. After 24 hours, the copied network outperforms the other networks slightly. Because the networks are not yet converged, it isn't possible to say if this network outperforms the other networks after convergence.

6 Conclusion and Discussion

In this thesis, we have trained multiple neural networks to play Go on small Go boards. Based on the training process and the evaluations of these networks we will try and answer the research questions asked in Section 1.

Research question 1

What neural network architecture performs best on small go boards?
The performance of the neural networks with different number of residual blocks on the 5×5 Go board were almost similar. The loss curves of these networks also show very similar behavior. The only difference during training was that with less residual blocks, the training was slightly faster. Therefore for the 5×5 Go board, we would suggest to keep the number of residual blocks in the neural network small. In this thesis, the smallest number of residual blocks used was 5. For future research, it would be interesting to see if smaller numbers of residual blocks could reach the same performance even faster.
For the 7×7 Go board we trained 2 networks with a different numbers of residual blocks, 7 and 9. The 9 residual blocks network got stuck in a local minimum, and performed worse than the 7 residual blocks network. We don't know the performance of the 9 residual block network when it doesn't get stuck in a local minimum. This makes it difficult to say if one of these two options is better than the other. Further research and more runs are needed to answer what number of residual blocks performs best on a 7×7 Go board.

Research question 2

Is it possible to transfer knowledge gathered on smaller boards to networks training on bigger boards?
When comparing the 1 hour checkpoints and the 3 hour checkpoints of each network on the 7×7 and 9×9 Go boards, we see a pattern of the pretrained networks outperforming the baseline networks. This pattern can most logically be explained by the fact that these pretrained networks do use the knowledge they have gathered from the smaller Go boards, and therefore have an advantage over the baseline networks. Based on this observation, we conclude that it is possible to transfer knowledge from networks trained on smaller Go boards to networks training on bigger Go boards.

Research question 3

Which transfer learning method performs better on larger Go boards?
When looking at the results of the 7×7 Go board after 12 and 24 hours of training, the copied and expanded networks have similar performance. The copied network performed a bit better after 12 hours of training, and the expanded network performed better after 24 hours of training. Because these are only single checkpoints in the learning process, it is difficult to put hard conclusions

to these results. Also, the value loss curves in Figure 8b and 9b are not yet converged. For now, we would suspect similar performance for both the copy method and the expansion method on the 7×7 Go board. However, more runs and longer runs are needed for a stronger conclusion.
For the 9×9 Go board, we also see similar performance for the copied and expanded networks. In Figure 10b it is even more clear that the value loss is not yet converged. We come to the same conclusion as on the 7×7 Go board. We suspect similar performance of the copy method and the expansion method. Again, more and longer runs are needed for a stronger conclusion.

6.1 Conclusion

Does increasing the size of the Go board during training have a positive impact on the performance of a neural networks trained with reinforcement learning?
Taking all information from this thesis, and using the conclusions made in the research questions, we see some evidence to conclude that increasing the size of the Go board during training could have a positive impact on the performance of neural networks trained with reinforcement learning. In particular, in the early stages of learning, the pretrained networks outperform the baseline network consistently. To be sure about this, more training runs per network are needed.
On the 7×7 Go board and the 9×9 Go board, we couldn't train the neural networks until convergence. Therefore it is difficult to say something about the ultimate performance of these networks. For a conclusion on this topic, not only more training runs are needed, but also longer training runs are needed.

6.2 Limitations

Although the results of this thesis look promising, there are a few limitations to take note of. One of the biggest limitations of this thesis is that every neural network is only trained once. We didn't have access to more computing power, and thus we had to make choices. By repeating the experiments, you would get a better idea about the behavior of the loss curves. It would become clear, if the average of the loss curves of the pretrained networks are actually lower than the average of the loss curves of the baseline networks.
Another benefit of more computing power, would be to extend the training times of the networks on the 7×7 Go board and the 9×9 Go board. The loss curves for these experiments are not yet converged. After convergence you could say with more certainty which network has the best performance.
The last flaw caused by a finite computing budget, was not tuning all parameters to find the perfect setting for every experiment. For instance the training time used for pretraining was now set on 3 hours. When looking at the win percentages of the networks trained on the 5×5 Go board, it is difficult to say if the network is significantly better after 3 hours than after 1 hour. We also

did not experiment with different number of filters per convolutional layer. It is possible that with a higher number of filters the knowledge gets transferred even better.

6.3 Future work

Addressing these limitations would be a good follow up on this thesis. An other area that could lead to new insights, would be to use different techniques in transfer learning. For instance to freeze certain layers of the network. In this thesis, there were no special transfer learning techniques used.
If the best parameter configurations and the best transfer learning techniques are found, then the next step would be to train a network from a 5×5 Go board, all the way up to a 19×19 Go board.

References

[1] Louis V. Allis. "Searching for solutions in games and artificial intelligence". English. PhD thesis. Maastricht University, Jan. 1994. ISBN: 9090074880. DOI: 10.26481/dis.19940923la.

[2] Peter Auer, Paul Fischer, and Jyrki Kivinen. "Finite-time analysis of the multiarmed bandit problem". In: *Machine Learning*. 2002.

[3] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. "Curriculum learning". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML '09. Montreal, Quebec, Canada: Association for Computing Machinery, 2009, pages 41–48. ISBN: 9781605585161. DOI: 10.1145/1553374.1553380. URL: https://doi.org/10.1145/1553374.1553380.

[4] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter I. Cowling, Stephen Tavener, Diego Perez, Spyridon Samothrakis, Simon Colton, and et al. "A survey of Monte Carlo tree search methods". In: *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI* (2012).

[5] Alexandra Chronopoulou, Christos Baziotis, and Alexandros Potamianos. "An embarrassingly simple approach for transfer learning from pretrained language models". In: *arXiv preprint arXiv:1902.10547* (2019).

[6] Michael Hu. *Alpha Zero: A PyTorch implementation of DeepMind's AlphaZero agent*. Version 2.0.0. 2022. URL: https://github.com/michaelnny/alpha_zero.

[7] Minyoung Huh, Pulkit Agrawal, and Alexei A Efros. "What makes ImageNet good for transfer learning?" In: *arXiv preprint arXiv:1608.08614* (2016).

[8] Levente Kocsis and Csaba Szepesvári. "Bandit based Monte-Carlo Planning". In: *In: ECML-06. Number 4212 in LNCS*. Springer, 2006, pages 282–293.

[9] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. *Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey*. 2020. arXiv: 2003.04960 [cs.LG]. URL: https://arxiv.org/abs/2003.04960.

[10] Koen Oppenhuis. *Variants of Monte Carlo Tree Search in the game ColorShapeLinks*. 2022.

[11] Aske Plaat. *Deep Reinforcement Learning*. Springer Nature Singapore, 2022. ISBN: 9789811906381. DOI: 10.1007/978-981-19-0638-1. URL: http://dx.doi.org/10.1007/978-981-19-0638-1.

[12] Christopher D. Rosin. "Multi-armed bandits with episode context". In: *Annals of Mathematics and Artificial Intelligence* 61 (2011), pages 203–230. URL: https://api.semanticscholar.org/CorpusID:207081359.

[13] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pages 484–489. ISSN: 14764687. DOI: 10.1038/nature16961. URL: https://doi.org/10.1038/nature16961.

[14] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. 2017. arXiv: 1712.01815 [cs.AI]. URL: https://arxiv.org/abs/1712.01815.

[15] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. "Mastering the game of Go without human knowledge". In: *Nature* 550 (Oct. 2017), pages 354–. URL: http://dx.doi.org/10.1038/nature24270.

[16] SURF Support. URL: https://servicedesk.surf.nl/wiki/spaces/WIKI/pages/30660184/Snellius.

[17] John Tromp and Gunnar Farnebäck. *Combinatorics of Go*. 2016. URL: https://tromp.github.io/go/legal.html.

[18] Erik van der Werf, H. Jaap van den Herik, and Jos Uiterwijk. "Solving Go On Small Boards". In: *ICGA journal* 26 (Oct. 2003). DOI: 10.3233/ICG-2003-26205.