UNIVERSITÉ PARIS DAUPHINE - PSL

Deep Learning project : Building a neural network for go playing

Martin LE DIGABEL
Baptiste LEFORT

Contents

1 Introduction

The goal of the project was to build a deep neural network that could learn the best strategy for playing the game of go. The main idea was using the computer vision to process the board of the game as an image. The convolution method was used and we tried several different implementations that derived from them. We have favored these methods after reading works on the subject. Computer vision has become the most successful method of learning board games then it naturally appeared as a promising method, and our results will show it. Also, the performances of a neural network were evaluated on two metrics : a mean squared error (MSE) and a binary cross entropy. We had two metrics since the output of the model was two heads. One output to get if the action taken was the best one and the other one to determine the prediction of the outcome of the game. One of the main constraints on this project was the number of trainable parameters that could not exceed 100 000 parameters. This constraint was set in order to keep a fair environment for all the students. Indeed, if we increase by a lot the number of parameters the performances could be better but not all the students could afford this computing power.

The starting point of the project was the basic neural network provided as a baseline. It had 10 convolution layers of 32 channels of size 19 by 19 for each. Each layer was around 9000 trainable parameters. After this convolutions, two flatten layers were used to train a small dense network that had two heads. We will quickly address this point later in the report.

After that, we found out that a ResNet would be better than a simple convolution network. It was indeed the case since the residual connection enables us to increase the number of parameters with less over fitting. We will address the design of the network later in the report.

After reading [1], we built a MobileNet with depth convolution wise layers that could learn better than the other neural networks. An entire part of the report will be dedicated to the hyper parameters used to compile the network.

Finally our researches shown a kind of architecture derived from the MobileNet: the ShuffleNet. We will address a last part to this model and its performance on the game of Go.

Before addressing the design of the experienced neural network and the research path and intuition we had, we will introduce the environment provided and more especially the data.

2 The data and the experimentation environment

In order to focus on the design of the neural networks, an entire environment of validation was provided. The input data is a randomly initialized numpy array. It contains only 0 or 1 for black and white pieces. The size of each array is 19 by 19 since it is the size of a real go board. This setup is generated 10 000 times. This will serve generating our training data.

We built a repository that had this structure :

1. utils: contains the training and recompile model files
2. trained_model: contains the .h5 files that are the weights of the trained models
3. results_training: contains all the .txt files that contain the loss of the networks we trained (used for visualisation purposes)
4. architectures: contains the python file of the implementations of the neural networks we built

```
GO_Project
+-- architectures/
| +-- baseline.py
| +-- resnet.py
| +-- ...
+-- trained_models/
| +-- baseline.h5
| +-- resnet.h5
| +-- ...
+-- results_training/
| +-- baseline.txt
| +-- resnet.txt
| +-- ...
+-- utils/
| +-- train.py
| +-- recompile_model.py
| +-- compare_models.py
| +-- show_model.py
```

We had to organize the project this way in order to collaborate more easily and because of the Google Colab environment. The central node of all these files was a notebook. All the python files were called in a sequential order to first select the implementation, train it and update compiler of the model (it will be address later in the report).

3 The training process: updating the learning rate

In [1], we could determine a proper training process for maximizing the learning capacity of the neural network. Indeed, we tried several optimizers such as stochastic gradient descent (which was the most efficient) and the adam. Both of them needed as a parameter the learning rate. We have chosen to adopt a non constant learning rate. Indeed, all the networks have been trained on 200 epochs for comparison purposes. The first 100 epochs we set the learning rate at 0.005 and we decrease after every 50 epochs the learning rate by dividing it by 10. Indeed, by adopting this strategy we could improve the performances with a larger number of epochs. If the learning rate would be constant at some point we would miss the optimal point.

4 The baseline : the simple convolution neural network

Firstly, we focused on getting a baseline in order to compare the neural network and have and idea of the minimum required performance. The architecture of the network was only 2D convolution layers and the two heads. The architecture of the heads were different and it's important to detail them.

1. The policy head: one 2D convolution layer with a kernel of size one, one filter and a relu activation. After one flatten layer and one activation layer with a softmax. We emphasize here that no dense layer was used.

2. The value head : one 2D convolution layer with a kernel of size one, one filter and a relu activation. After one flatten layer and two dense layers with the last one that has one neuron with a sigmoid activation (for output only values between 0 or 1)

On both of the heads a L2 regularization was applied. The reason is to prevent from overfitting by minimizing the weight of the largest values.

Before the heads, 10 2D convolution layers were mounted. We have chosen this number of layers such that we were the closest to the maximum number of trainable parameters allowed. All the convolution layers had 31 filters and a kernel size of 3x3 except the first one that had a kernel size of 1x1. Here is the graphical representation of the neural network.

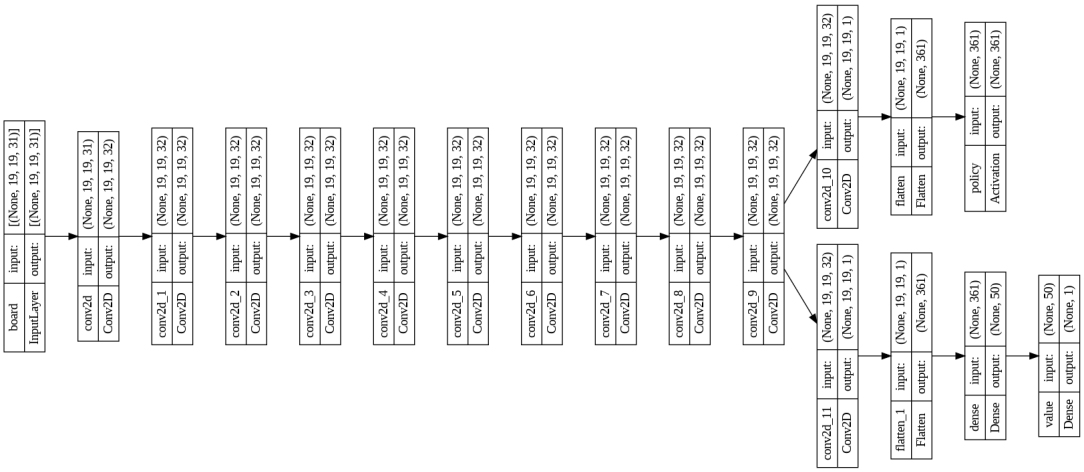

Figure 1 - Baseline NN architecture

The total number of trainable parameters is 102 471.

This architecture was already quite efficient and seemed to be a good starting point. In the report we will compare the models with a focus on the policy accuracy. For the baseline network with a stochastic gradient descent, here is the performances for 200 epochs (with the α divided by 10 at the 100th iteration and at the 150th iteration).

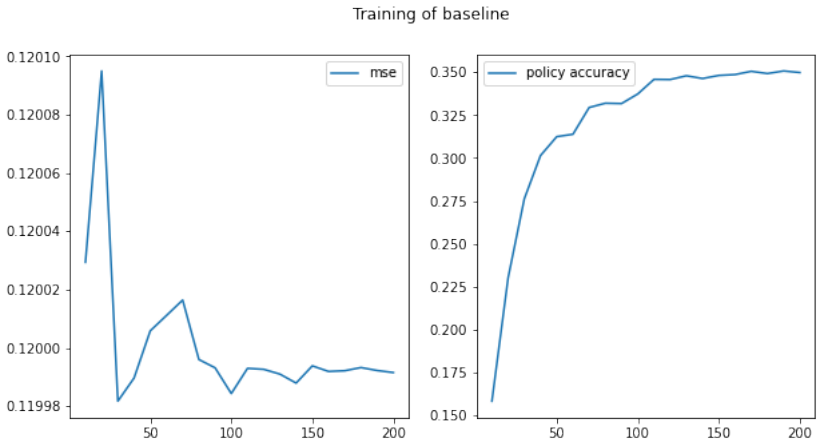

Figure 2 - Baseline NN architecture metrics

We can see above that the network keeps learning better thanks to the changes of the learning rates. With more epochs the policy accuracy would have kept improving to nearly 0.4 (we only plot for 200 epochs in order to compare with the other networks). The fact that the amount of data is very large allows us to consider that the network will not over fit, using the getBatch function implemented in the golois file.

Finally, we see above that the MSE oscillates before converging. It shows us that the number of epochs should be at least greater that 150. The policy accuracy also validates the training method with the adaptive learning rate by keeping improving when the learning rate changes.

5 The ResNet

The second neural network that we built was a resnet. Using a residual connection is largely advised in the literature. The residual connection adds the output of one layer to the input of a later layer in the network, allowing the network to learn a residual mapping that predicts the difference between the input and output. One of the interests of the residual connection is that it enables to build deeper network without losing information. First, we set the number of filters to 31 for each of the n residual blocks. Then, we decided to test two setups. One with many blocks where each convolution layer has few filters and the other with few blocks but many filters. The aim of these test is check if metrics are really sensitive to the number of blocks. Finally, we tried an architecture with a non-constant number of filters inspired by auto encoders models. Let's have a $C = 31$ filters. We split our architecture in 3 parts: First part has $2C$ filters, the bottleneck part has $\frac{1}{4}C$ filters and the third part C filters.

5.1 ResNet - 31 filters - 4 blocks

First, below are our results for ResNet network with constant number of filters:

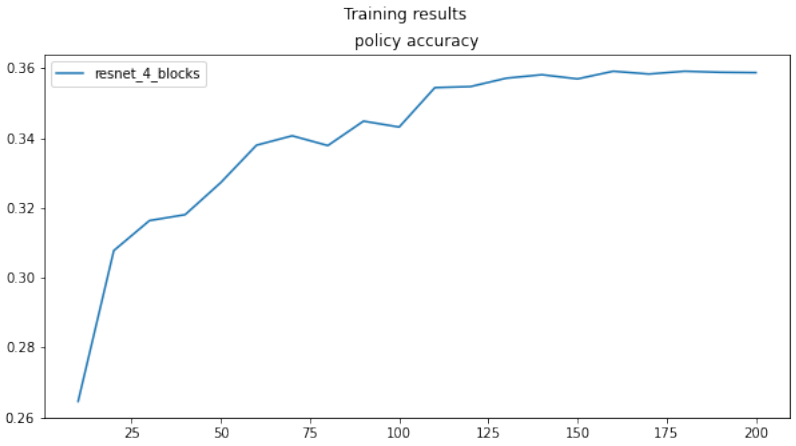

Figure 3 - ResNet policy accuracy 31 filters - 4 blocks

As we can see on the graph it doesn't show better results than the baseline. A good point may be the better stability of our results.

5.2 ResNet - Number of blocks comparison

Reading the article [1] we tried to use more blocks in our ResNet architecture to exploit its strength in deeper networks. We made 3 tests of models with respectively 4, 6 and 8 residual blocks with adapted constant number of filters to respect the constraint of the 100 000 parameters max. Unfortunately the accuracy isn't better as our results show it below:

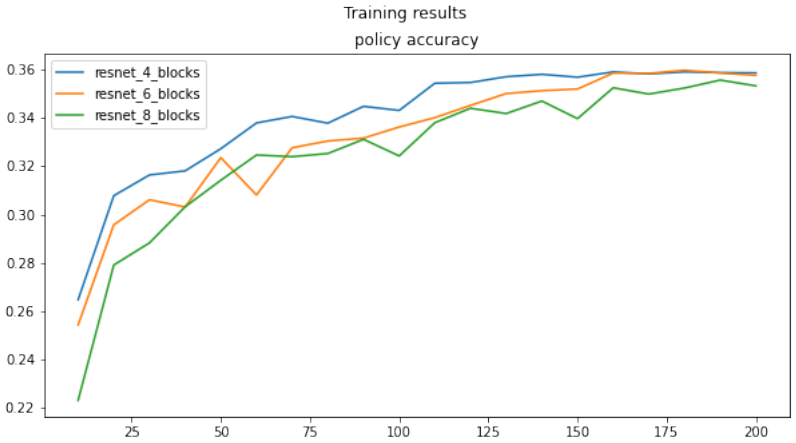

Figure 4 - ResNet number of blocks comparison

5.3 ResNet - encoder architecture - 4 blocks

An experiment we made has been to keep the ResNet global architecture and try to reproduce a kind of auto encoder by changing the number of filters in the network and create a "bottleneck" at his middle. We believed that in any case it would give us more information on the sensitivity of our results on the non-consistency of the filters in the residual blocks.

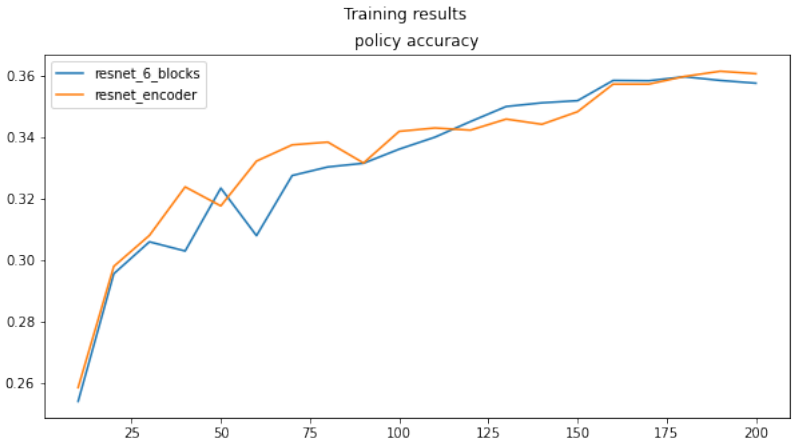

Figure 5 - ResNet encorder architecture 4 blocks

We can observe that our metrics show pretty same results after 200 iterations (despite the MSE seems different it's only due to the scale of the graph). The resnet_encoder took a longer time to learn but showed even slightly better results after 200 iterations. Because the performance differences between the two architectures are not significantly different we will keep a constant number of filter in the following sections for a clearer comparison of the architectures.

6 The MobileNet

We decided to implement a MobileNet. In [1], it is shown that it is the most efficient network for go playing. It is also our conclusion when we compared the results of the metrics and more significantly the MSE loss. We will firstly explain the various reflections that have been carried out on the subject. The principle of the MobileNet is to derive a ResNet by switching the classic 2D convolution layer by a depth-wise convolution. It enables us to use more blocks since this layer has less trainable parameters. For the implementation we have used what is shared in [1] with the trunks method. We will not re-explain how the MobileNet works here and we only focus on the results.

We have tried two types of MobileNet. One with few blocks and many filters and the other with few filters and many block. Although the difference is not significant, we can conclude that the one with many block is more efficient especially since it keeps learning after 200 epochs (as we will show). This result is not surprising since on the main interest of the residual block is to generate deep network that can keep learning without loosing information. Below is a detail of the two networks parameters that we tried.

1. More blocks and less filters : 20 blocks, 31 filters and 64 trunks
2. Lore filters and less blocks : 10 blocks, 62 filters and 128 trunks

The main idea behind these implementation is to validate that with a maximum number that is quite low the it is more interesting to have many blocks. Indeed, residual blocks are interesting for making the network deeper without over fitting or gradient vanishing/exploding but with a limit at 100 000, maybe it could lost its interest.

The architecture of a block is the same for both of the network and is plot below.

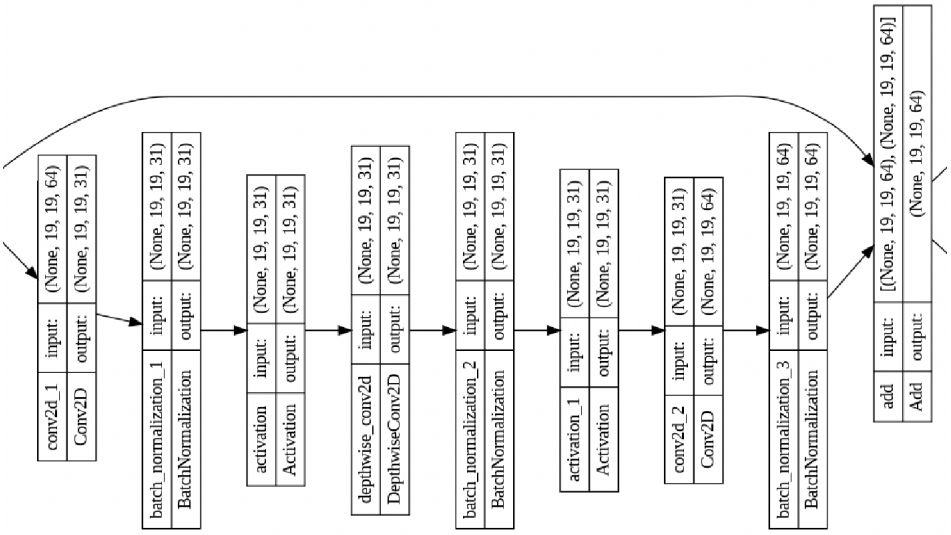

Figure 6 - MobileNet's block architecture

In total, we had 100 980 parameters. The training schema was the same as described in the third section. We can see below how the network has learnt.

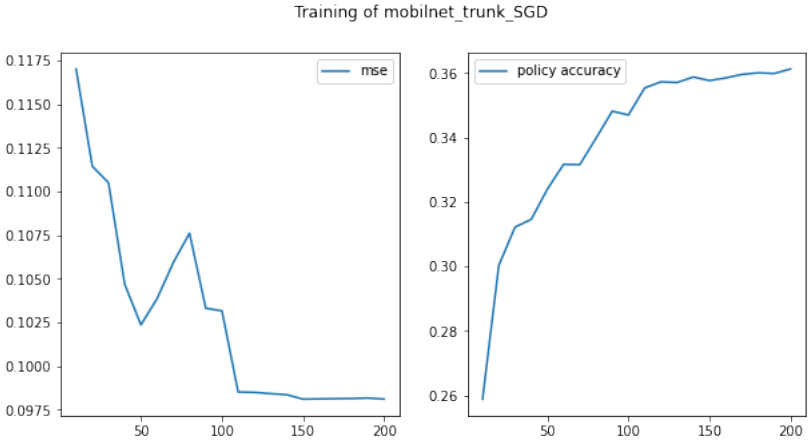

Figure 7 - MobileNet performances with 20 blocks - 31 filters and 64 trunks

We can even see that at 200 epochs the network could still keep learning. Also, the loss is the best we could obtain compared to all the other networks. It reached a minimum that is quite constant around 0.0975. By decreasing more the learning rate, we could keep hoping for decreasing the loss. As a comparison, below is plotted the results of the network with many layers and few blocks.

Figure 8 - MobileNet performances with 10 blocks - 62 filters and 128 trunks

The accuracy seemed to have reached a convergence and the loss as well. Also, we can see that the loss is less stable than for the first architecture especially for the first epochs. We have decided to keep going with the first architecture since it seemed to be more promising.

7 ShuffleNet

Finally we implemented a ShuffleNet after reading [3]. It is described as a CNN efficient architecture designed especially for devices with very limited computing power. Moreover use cases show that ShuffleNet maintains a good accuracy with a lower computational cost compared to classic CNN and ResNet architectures. This is reached by the introduction of two tools: point-wise group convolution and channel shuffle. In our case where we dispose of 100 000 parameters maximum this kind of model seemed to make sense. Also, it came to our mind since we were exploring residual networks.

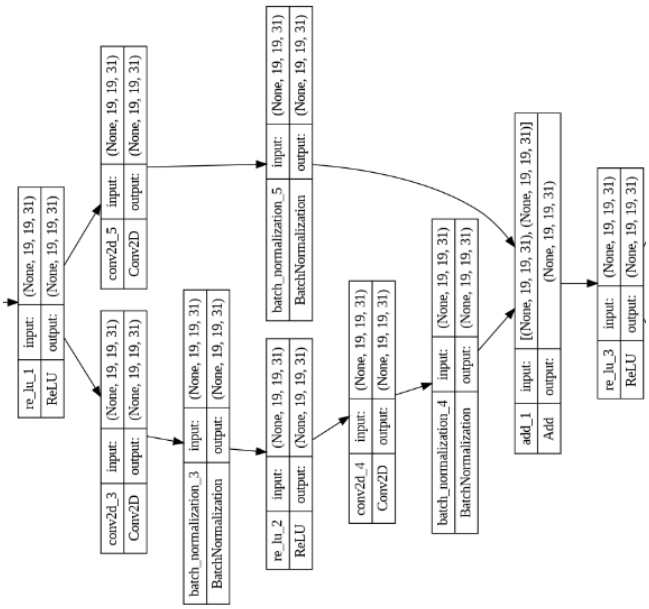

Figure 9 - ShuffleNet's block architecture

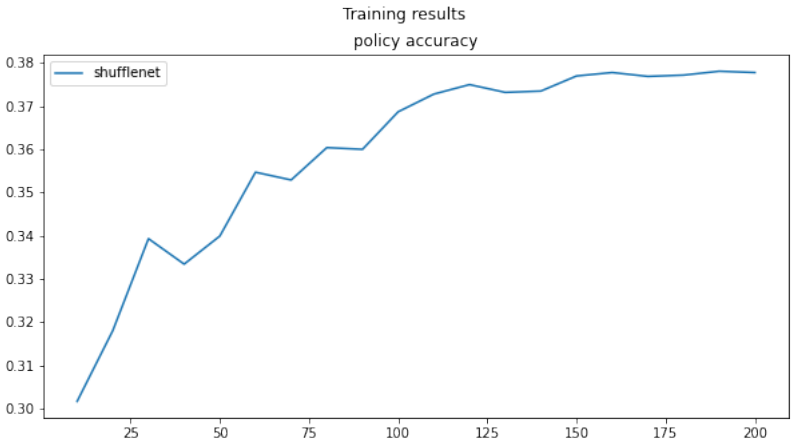

Figure 10 - ShuffleNet's performances

This network had 98 875 trainable parameters.

The shuffleNet was convincing since it reached the highest policy accuracy compared to the other networks. It would be also very interesting if we could increase the number of trainable parameters

and more especially by increasing the number of blocks. Indeed, even though this network could over fit faster than a ResNet for example, the data processing in the residual connection seems to be interesting to the network.

8 Final comparison of the models

All our work was based on comparing our experimentation to our baseline. Firstly, we ruled that this baseline was good enough to guide our work. Since we saw that the right direction seemed to be the residual net after some discussions with our classmates and professor, we could confirm that building residual networks were always better than simple convolutional network. Below is plotted all the policies performances of the models we experimented.

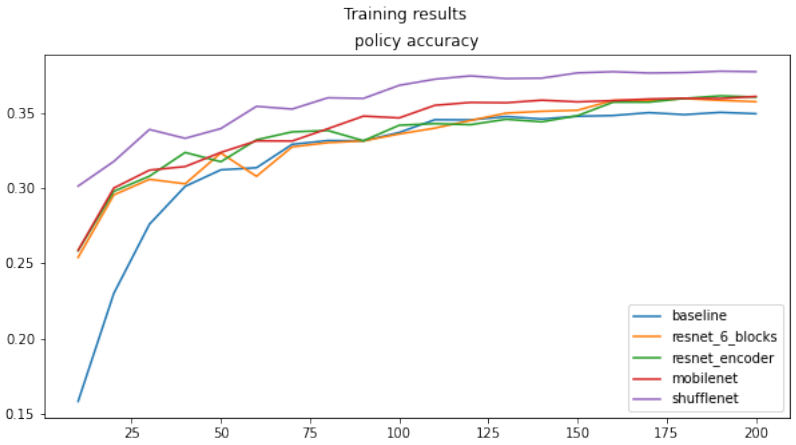

Figure 11 - Comparison between all of the architectures

As seen on the graph ShuffleNet shows a promising performance and confirm what we read about it. Despite the small number of parameters we used it proved to be efficient. Also, the execution time was quite the same for all the networks using the colab environment with the GPU.

9 Conclusion

We trained different convolutional architectures for computer vision: Full CNN, ResNet, MobileNet and ShuffleNet. We plotted and described each experiment we led for each of this models. Finally we noted similar results as the article [1] we mainly focused on.

ResNet and MobileNet have outperformed our baseline composed of only Convolutional layers. Our last model ShuffleNet showed the best policy of our study and then is revealed to be the best model (as we noted the policy to be the most important metric for the game of Go).

We can emphasize that we made the strong hypothesis that 200 iterations were enough to make meaningful comparisons of our models. We trained our networks we wondered if our choice to set an alpha at 5×10^{-3} and then divide this alpha by 10 at the iteration 100 and 150 was the optimal thing to do considering that our models may not learn at the same speed. But being limited by our material (we used Colab which limits the access to the GPU) we thought that 200 iterations was good enough to compare our models. We could find the optimal step for the learning rate

References

[1] Tristan Cazenave, *Mobile Networks for Computer Go*, LAMSADE, Universite Paris-Dauphine, PSL, CNRS, Paris, France, 2022.

[2] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, Jian Sun*ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices*, 2017

[3] 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, Demis Hassabis*Mastering the Game of Go without Human Knowledge*, 2017