Deep Learning for Go Game

Abdelkader Chihab Benamara

January 23, 2022

Abstract

This paper is a summary for the Deep Learning course which was based on training an artificial neural network able to play the game of GO, we will try to give a comparison between several architectures which all are based on producing a prediction for the two possible outputs (policy) and (value) for a given board state as input, at the end of our analysis we can clearly good performances for models based on SE[1] compared to other basic residual models or those having attention modules.

1 Acknowledgement

In order to run and train our models for this project we have based this with the help of clusters of **Grid5000**[2] which has an important number of computational resources based at various locations around France and Luxembourg, during this project i have worked basically on a server located at Lyon, France. This cluster has 2 nodes with **512GB** of RAM and **8x Nvidia Tesla V100 (32 GiB)** GPU's, this configuration was helpful to give us the possibility of working on **1 million** games dataset and also to train for much more epochs than we were doing for the first few weeks on Google Colab platform.

2 Base Model Architecture

For purposes of reusing the code without repeating a ton of lines we decided to organize the code and to have a base model which starting from it we can easily change the internal architecture of newer versions, this base model we called it **DGM** and so on the incoming version will inherit from this basic class and can include the specific changes for each model, in this section we will see in details how we made this DGM class.

As the input of our network doesn't change we have a method for the input block which take a **19x19x31** board state representation (31 being the number of planes). The following figure summarize this structure :

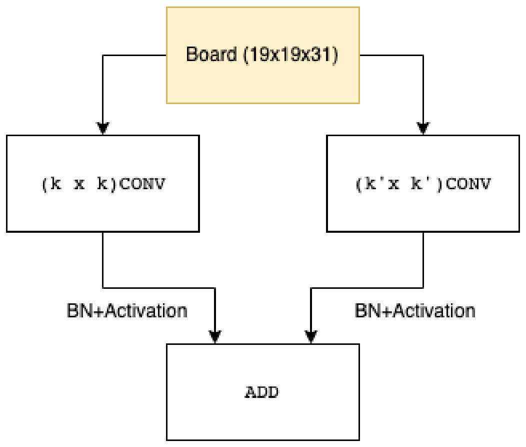

Figure 1

Input Block architecture that takes 19x19x31 input boards and sum up two convolutions of different sizes k and k'.

The next part of these networks is the body block and here which come the modification and the specifics of each model this block is an abstract implementation which can be redefined for each future version and we will talk in much more details about this part in the next sections.

We have also a fixed part of our model which is the outputs blocks let us start with the policy head which produce a probability distribution over the **361** squares of the board.The following figure illustrate the structure of this block :

[1]SE : (**S**)queeze-(**E**)xcite blocks

[2]Grid5000 : https://www.grid5000.fr/

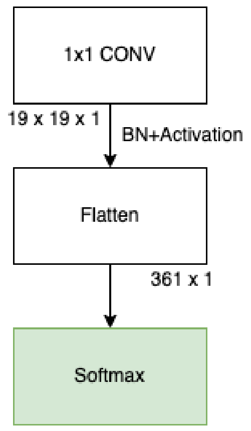

Figure 2
Output Block architecture (Policy head) that takes $19 \times 19 \times q$ input shapes and give a **361** probabilities of each square of the board.

We have another head as we have already said in the beginning which is the value head, this output return a value between 0 and 1 in order to predict the winner starting from a given input board state. The following figure illustrate its architecture :

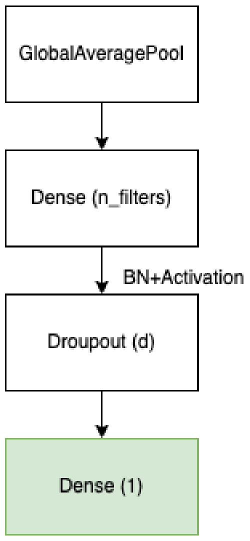

Figure 3
Output Block architecture (Value head) that takes $19 \times 19 \times q$ input shapes and give a **1** probabilities value between 0 and 1 predicting the winner.

Note that the parameters $n\_filters$ and d are an attributes that we have tuned a lot and we will set them at the end for our final model and they do change with each different architecture.

3 First Attempts

For the first three weeks of the tournament organized by Mr.T Cazenave[3] in which I trained a basic Residual Neural Networks with **SE** at each block of this configuration and we tried to find the best trade-off (width-depth) of these several models the best we could do was a **6** residual blocks with **SE** having **64** filters each and a squeeze ratio of **0.25** a basic block can be represented by the following figure :

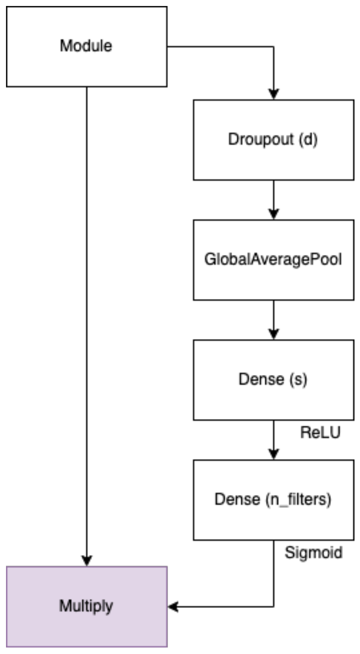

Figure 4
SE Block architecture , which is included inside each module block (body of the network) with dropout d to reduce overfitting and s being the number of filters for squeeze part

We have trained this architecture for 500 epochs and it did well in terms of performance with over 50% of policy accuracy at validation, even-thought this performance the main issue with the residual blocks is that we cannot make the network deeper as we are limited to **1M** parameters and this was the motivation behind using smaller and lighter networks such that MobileNet and MixNet. We will see more details in the coming sections.

4 Final models

As already mentioned earlier the only part that differ in our implementation is the body of networks and in this section

[3]Tristan Cazenave https://www.lamsade.dauphine.fr/ cazenave/

we will talk more about the several architectures that we have implemented and trained.

4.1 Inception Network

The idea behind the inception block is to add several sizes of kernels in order to increase the number of channels and exploit the **31 planes** of the board state, a basic inception module that we have implemented have this following structure :

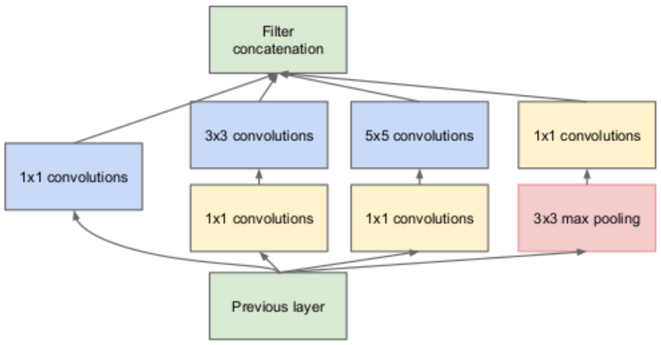

Figure 5
Inception Block architecture , which is included inside each module block (body of the network) with SE blocks before the last layer of this block – src JalFaizy Shaikh[4]

4.2 MobileNet Network

The idea behind the mobile block is using the inverted residual blocks which make the network lighter as we are limited to 1M parameters it is a good idea to have much deeper networks with and an important number of filters for the convolutions and this was possible with the help of MobileNet blocks. MobileNet 'bottleneck' blocks that we have implemented have this following structure :

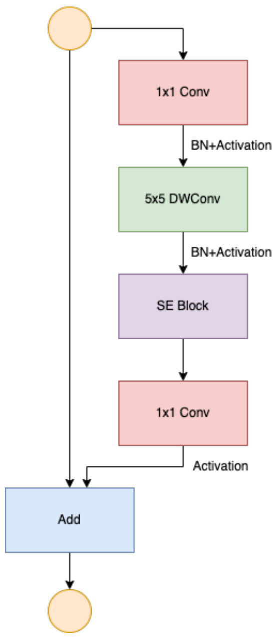

Figure 6
MobileNet Block architecture , which is included inside each module block (body of the network) with SE blocks before the last 1x1 Conv layer of this block, the activation has been changed during the training we will see that furthermore

4.3 Mnasnet Network

The idea behind the mnasnet block is moreover the same of the MobileNet architecture with using the inverted residual blocks but now with different size of kernels (mixed version) which make the network lighter as we are limited to 1M parameters it is a good idea to have much deeper networks with and an important number of filters for the convolutions and this was possible with the help of MnasNet blocks. MnasNet blocks that we have implemented have this following structure :

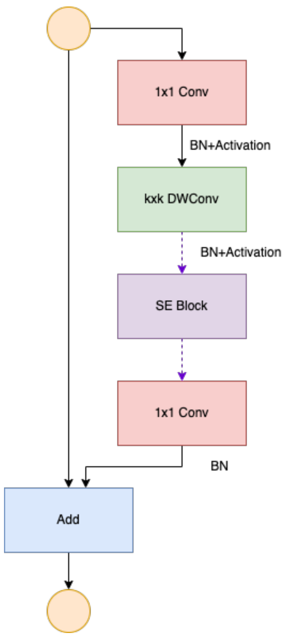

Figure 7
MnasNet Block architecture , which is included inside each module block (body of the network) with an **optional** SE block before the last 1x1 Conv layer of this block, the activation has been changed during the training we will see that furthermore

At the beginning of the network's body we have a new 5×5 Separated Convolution which has the following architecture :

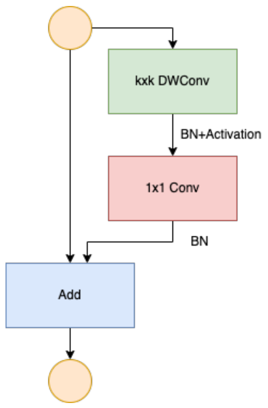

Figure 8
SepConv Block architecture , which is included after the input block of each network

5 Experiments & Results

In this section we will focus more on the phase concerning the training and evaluation of this three models we have selected to be on the top 3 for this task of playing the Go Game.

All this training have been done using the same optimizer (**Adam**) but with different maximal learning rate fir each network and in order to have a decreasing sequence of learning rate we opted for the cosine annealing decrease which can have the following expression to calculate the learning rate at each epoch k :

$$\eta_k = \eta_{min} + \frac{1}{2}\left(\eta_{max} - \eta_{min}\right)\left(1 + \cos\left(\pi \times (\frac{k}{N})\right)\right)$$

Where here N is the number of epochs and $\eta_{min/max}$ are the min/max values that can the learning rate take and during our experiments we have noticed that whenever we change the η_{max} we got improvement in the performance. The following graph show how the decrease of the learning rate is working during the training epochs :

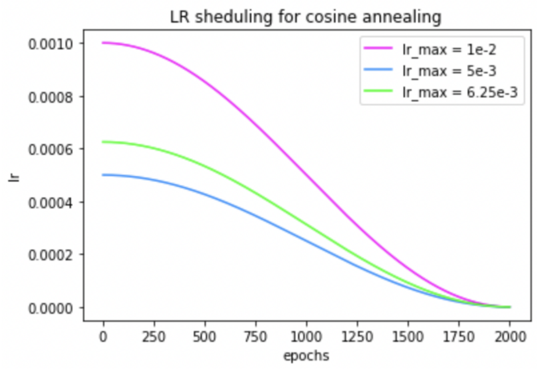

Figure 9
The decreasing sequence of learning rates using cosine annealing without restart warm-up (just one cycle) in function of lr_max parameter

In order to compare this three models we have trained them for 2000 epochs and we have got the following results :

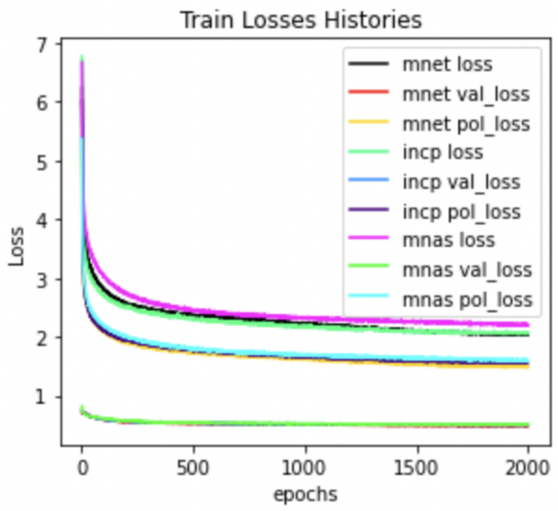

Figure 10
Histories of losses comparing MobileNet vs InceptionNet vs MnasNet

We can notice nearly the same performances for these three networks which is a good news as we are supposed to compare only the best models, but from this graphs it is a little bit complicated to take a conclusion on which model is better of our task, moreover we can notice how the inception is doing well against this two mobile architectures, we will see next the specific output head to get more results that can be interpreted.

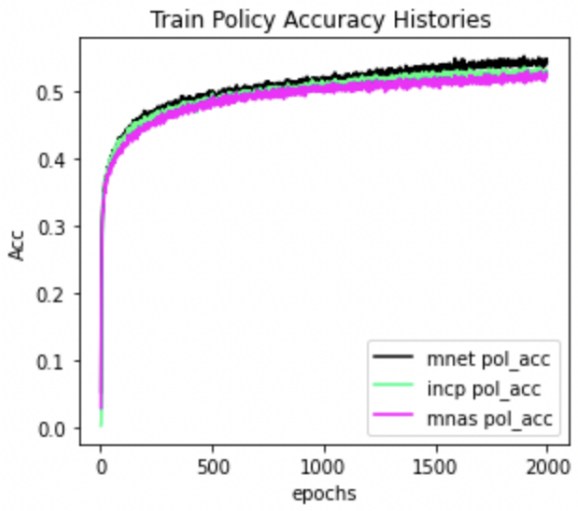

Figure 11
Histories of policy head accuracy comparing MobileNet vs InceptionNet vs MnasNet

We can notice now how the policy accuracy is better for the MobileNet especially for epochs starting from 1000 and inception model still doing well against the MnasNet and have a good accuracy , after 2000 epochs he have around the **55%** for MobileNet and Inception and **54%** for MnasNet, note that for the first 250 epochs inception was taking the lead over the other networks.

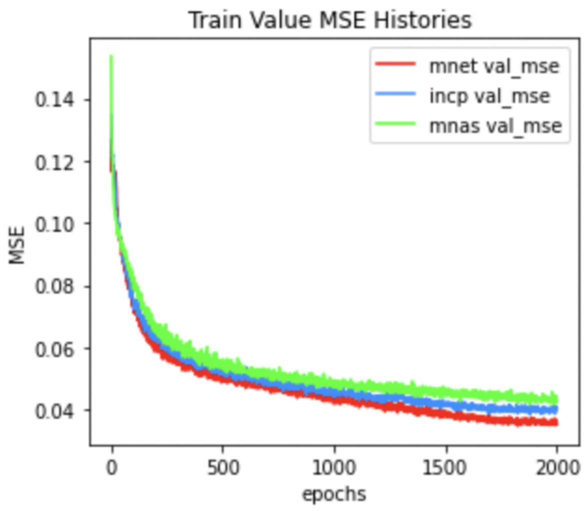

Figure 12
Histories of value head MSE comparing MobileNet vs InceptionNet vs MnasNet

The same observation can be noticed for the value head where in the previous plot we have the MSE (Mean Squared Error) for the value head and we can clearly see how the MobileNet is taking the lead and is converging to much more smaller values of MSE starting from epoch 1250 and we still have inception model in the second position.

5.1 Performance of best model

As we have already said in the previous section the MobileNet took the lead over the other architectures so we decided to select this network and try to train several versions of it (by changing and tuning the number of filters) and the depth (number of blocks), by doing that we reported the following performances after 7500 epochs of training on the 1M games DataSet from KataGo[5]

Model	Policy	MSE	Loss	# Params
mnet_256_21	57.12%	0.0291	1.88	1.003.869
mnet_235_23	57.05%	0.0293	1.89	1.003.154
mnet_302_18	56.90%	0.0299	1.90	1.006.669
mnet_256_22	56.80%	0.0297	1.90	1.005.245
incp_192_14	56.35%	0.0297	1.91	1.000.000

[5]KataGo : https://katagotraining.org/games/

Note that in all this models we have used the swish[6] activation function and in the naming convention *model_n_d* where model is the name of our architecture and *n* is the number of filters in each block and *d* is the number of blocks.
From the above results we can choose the winning model to be the selected one on the table with 21 bottleneck (with SE blocks inside) and each with 256 filters, note that the loss is the sum of value and policy head losses as we kept unchanged the value weight at the beginning of this project we tried to increase the value head weight by setting it to **16** and keeping the policy head weight to **1** in order to avoid that during the training we give more importance to the policy head compared to the value but it doesn't do better in our case as we had only **55%** (compared to **57%**) while changing the value head weight to 16 for the same architecture and the same hyper-parameters.

6 Conclusion

At the end of this study we finally decided to retain the winning model (MobileNet with 21 bottleneck block and 256 filters at each) with the swish activation function and trained using the Adam optimizer with cosine annealing for the learning rate scheduling without restart warm up and it does give a pretty good results with over **57%** accuracy for the policy head at validation and **0.0291** MSE for the value head.

References

[1] T. Cazenave, "Mobile Networks for Computer Go," *IEEE Transactions on Games*, pp. 1–9, 2020.

[2] T. Cazenave, "Cosine annealing, mixnet and swish activation for computer Go," pp. 1–8, 2021.

[3] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-Aware Neural Architecture Search for Mobile," *arXiv:1807.11626 [cs]*, May 2019. arXiv: 1807.11626.

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," *arXiv:1409.4842 [cs]*, Sept. 2014. arXiv: 1409.4842.

[5] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv:1704.04861 [cs]*, Apr. 2017. arXiv: 1704.04861.

[6] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for MobileNetV3," *arXiv:1905.02244 [cs]*, Nov. 2019. arXiv: 1905.02244.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv:1512.03385 [cs]*, Dec. 2015. arXiv: 1512.03385.

[8] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-Excitation Networks," *arXiv:1709.01507 [cs]*, May 2019. arXiv: 1709.01507.

[9] I. Loshchilov and F. Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts," *arXiv:1608.03983 [cs, math]*, May 2017. arXiv: 1608.03983.

[10] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, "Don't Decay the Learning Rate, Increase the Batch Size," *arXiv:1711.00489 [cs, stat]*, Feb. 2018. arXiv: 1711.00489.

[11] J. Barratt and C. Pan, "Playing Go without Game Tree Search Using Convolutional Neural Networks," *arXiv:1907.04658 [cs]*, July 2019. arXiv: 1907.04658.

[12] T. Cazenave, "Residual Networks for Computer Go," *IEEE Transactions on Games*, vol. 10, pp. 107–110, Mar. 2018.

[6]swish : $x \mapsto x \times \sigma(x)$