Dauphine | PSL
UNIVERSITÉ PARIS

Deep Learning Project

M. Tristan CAZENAVE - Master 2 IASD

# Deep Neural Networks for Computer Go: Report

*Authors:*
Élise CHIN
Noé LALLOUET


February 12, 2022

# Contents



# 1 Introduction

The aim of this project is to build and train a neural network with the goal of playing computer Go.

## 1.1 History of Computer Go

Go is a strategy board game that originated in China more than two thousand years ago. A two-player game, Go's aim is for one to control a larger territory than one's opponent, by alternatively placing pawns on a board.

As artificial intelligence became a forefront topic in research, many scientists took on the challenge of beating top human players at games. The first games to be optimized were simple games such as Tic-Tac-Toe, but more sophisticated games quickly followed, as reigning chess champion Garry Kasparov infamously fell against chess engine DeepBlue, in a 1997 match that drew the world's attention on the potential of artificial intelligence.

However, human Go players continuously resisted AI's assaults, due to the sheer number of possible game states that renders minimax-like algorithms impossible to compute. It was only in 2016 that Google's DeepMind cracked Go by using a revolutionary deep reinforcement learning strategy [1].

## 1.2 The project

Modern computer Go architectures use Monte-Carlo Tree Search (MCTS) combined with a two-headed neural network that is used to predict the optimal policy and the estimated value of a given position.

The goal of the project is to train such a neural network, which will then be plugged into a MCTS program and ran against competitors in weekly contests. The deep learning framework used for the project is Keras. Students are free to use an architecture of their choosing, but all models will be constrained by an upper bound on the number of network parameters, set at one million. Students are expected to provide a trained model accompanied by a written report for grading.

# 2 Achitectures and techniques

In this section, we shall present the different architectures and techniques that have inspired us during the project's duration.

## 2.1 ResNet

The base model provided by Prof. Cazenave is a deep residual convolutional neural network, which we shall refer to as ResNet. It has been shown that residual networks perform better than their non-residual counterparts, be it in computer vision [2] or computer Go [3].

Residual networks introduce a novel idea to the world of deep convolutional neural networks: the residual block.

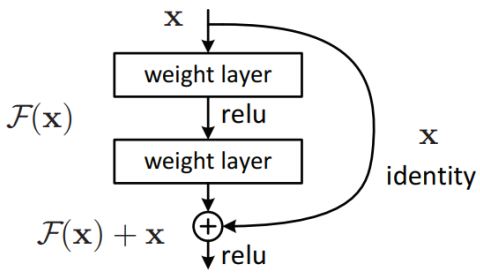


**Figure 1:** The residual block

As we can see in Figure 1, a residual block is obtained by adding the output of a hidden layer to the unchanged input. As such, some of the original input is allowed to pass through a large number of layers. A residual block achieves two important goals in the training of a

deep network. First, it addresses the vanishing gradient problem. As a matter of fact, being able to access the input of a block allows the network to backpropagate gradients in an easier fashion without having to fully propagate through complicated operations such as convolutions. Second, it allows the neural network to trivially learn the identity function, which was identified as a problem by prior research.

Since their creation, ResNets have become a staple of the computer vision scientist's arsenal, due to their high learning capability. Until very recently, ResNets have held the top spots on all major image recognition competitions. Residual networks' great performances provide a good baseline on which to try to improve a neural network designed for computer Go.

## 2.2 MobileNets

### 2.2.1 MobileNetV1

MobileNets [4], introduced by researchers from Google in 2017, may be considered as a small revolution for computer vision. This novel architecture drastically reduces the number of parameters of convolutional neural networks without noticeably dropping in performance. It has enabled new perspectives of applications of deep CNNs, from embedded systems to mobile phones. The crucial innovation behind MobileNet is the use of depthwise convolutions instead of regular convolutions. Depthwise convolutions work by seperating filters throughout the input tensor's channels, allowing for much more lightweight computation. Then, these channel-wise convolutions are stacked and convolved with a 1x1 kernel in order to attain the target channel number.

### 2.2.2 MobileNetV2

MobileNetv2 [5], built by the same team that created the MobileNet, in a bid to improve their model's performance, introduces a new building block named inverted residuals. Acting as a replacement for the separable depthwise convolution of MobileNet, the inverted residual takes the shape of a bottleneck. In the residual block, the input tensor is depthwisely convolved to a smaller dimension that contains all the necessary information. Then, the result of these operations is convolved to a larger size, called expansion, and added to the input in regular ResNet-like fashion. These novelties brought a large increase in performance of more than two points on image classification tasks from the original MobileNet, while reducing the number of parameters.

### 2.2.3 MobileNetV3

MobileNetV3 [6], introduced in 2019, are the latest evolution of Mobile Networks. Created using a Network Architecture Search (NAS) algorithm, they use yet another new building block introduced by [7], the squeeze-excitation (SE) block. SE blocks are defined by the aggregation of a tensor's feature maps (squeeze) into a representation of inter-channel dependencies, that is then multiplied to the original input. As such, SE blocks in MobileNetV3 bring lightweight channel-wise attention modules to the bottleneck structure in order to capture channel information. MobileNetV3 also introduces a new non-linearity, called h-swish, which is a modification of the swish activation function ($swish(x) = x \cdot \sigma(x)$). h-swish replaces the expensive sigmoid function by a scaled ReLU6 function, which reduces inference time.

## 2.3 CBAM: Convolutional Block Attention Module

Published in 2018, [8] introduces a building block for convolutional neural networks that uses attention. Whilst squeeze-excitation blocks may be regarded to as channel-wise attention modules, CBAM also brings attention to the spatial dimension. The proposed attention module separates channel attention and spatial attention. Channel attention is computed by using the results of max pooling operations and average pooling operations in a multi-layer perceptron, before using an activation function. Spatial attention also uses pooling, but this time along the channel dimension. Then, the output of the pooling is convolved and activated. Finally, these two submodules are sequentially multiplied to the original input to complete the CBAM block. A visual explanation of CBAM shall be found in Figure 2.

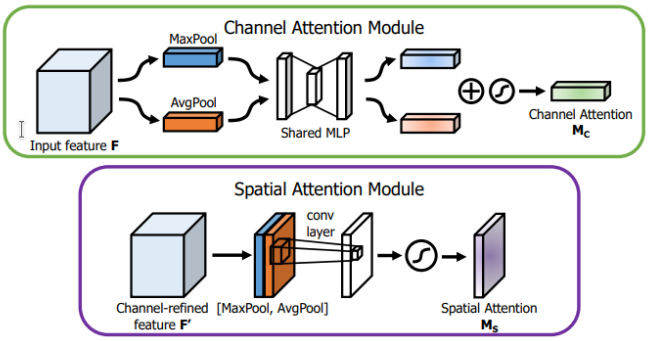


**Figure 2:** The two attention submodules of CBAM

### 2.4 Mixed Convolutions

Mixed convolutions originate from a research effort led by Tan Mingxing and Le Quoc V. that aimed to study the impact of different kernel sizes on the performances of deep convolutional neural networks. Based on this effort, the authors introduce [9] a novel type of convolution : Mixed convolutions. The main idea behind mixed convolutions is to take advantage of the performance increase of different kernel sizes without the jump in number of parameters associated. To do so, mixed convolutions separate the input in N groups along the channel axis. Each of these N groups will then be submitted to a convolution operation with a different kernel size. A popular choice for kernel sizes for mixed convolutions are 3x3, 5x5 and 7x7. Mixed convolutions have been proven to be effective in computer Go [10].

### 2.5 EfficientNet

Finally, the last architecture we investigate is EfficientNet [11]. EfficientNet architectures stem from the observation that, although numerous attempts at increasing model performance through network depth, filter depth and image resolution have been successful, no effort had been made to integrate all three scaling components harmoniously. As such, the authors propose a scaling coefficient $\phi$ that rules scaling on all three aspects in the following fashion:

$$\begin{aligned} \text{depth } d &= \alpha^\phi \\ \text{width } w &= \beta^\phi \\ \text{resolution } r &= \gamma^\phi \\ \text{s.t } \alpha \cdot \beta^2 \cdot \gamma^2 &\approx 2 \\ \alpha \geq 1, \beta \geq 1, \gamma &\geq 1 \end{aligned}$$

EfficientNets have been, since their release, widely used in the community due to their extremely high performances, achieving state-of-the-art results on many tasks.

## 3 Experiments and results

### 3.1 First model: ResNet

Our first objective was to find a good set of hyperparameters to improve the performance of the base ResNet model. For that purpose, we kept the default parameters and experimented individually with different values of batch size, number of filters, optimizers (Adam, SGD), number of blocks and even the parameter `N`, the number of samples the network goes through at each iteration, while satisfying the one million trainable parameters constraint. All models were trained for 200 epochs. The default parameters were the following:

```
"N": 20000,
"num_blocks": 6,
"batch_size": 128,
"filters": 64,
"learning_rate": 0.005,
"policy_loss_weight": 1.0,
"value_loss_weight": 4.0
```

This trial and error process enables us to obtain an improvement of 0.4% between the worst performing model (ResNet model with 32 filters) and the best one (ResNet model with a batch size of 64). The final best ResNet model achieved a validation policy categorical accuracy of **41.1%** and a validaion value MSE of **0.077**.

| **Hyperparameter** | **Policy accuracy (%)** | **Value MSE** |
|---|---|---|
| Default | 40.9 | 0.083 |
| Batch size = 64 | **41.1** | 0.077 |
| Batch size = 256 | 40.6 | 0.08 |
| Adam(0.001) | 40.9 | 0.077 |
| N = 10000 | 40.8 | **0.075** |
| filters = 32 | 36.9 | 0.09 |
| filters = 74 | 38.4 | 0.085 |

### 3.2 Second model: MobileNet + SE + Swish

A review of the recent literature led us to select the MobileNetV3 architecture as our next experiment. After some investigations regarding the parameters to choose in order to capture as much information as possible, we settled on the following architecture: our MobileNet is composed of 13 bottleneck blocks (similar to the one presented in [12]), 384 expand filters and 64 trunk filters. To focus attention on more important channels, a SE block [13] is added before the last convolution and batch normalization of each bottleneck block. As proposed in [10], we used the Swish activation function, and scheduled the learning rate with Cosine Annealing from 0.01 to 1e-7. The model was trained with a batch size set to 64 and Adam optimizer for 1000 epochs first. Since the first run was not very conclusive – only a policy accuracy of 40% (Figure 4), we trained for another 1000 epochs with cosine restart. The model was not performing better after training for more epochs. The MobileNet with cosine annealing and a restart at the 1000th epoch reached a

validation policy categorical accuracy of **40.8%** and a validation value MSE of **0.078**. It unexpectedly performed worse than the ResNet model, which was a simpler model and which took less time to train.

One of the reasons that could explain this disappointing performance is perhaps the annealing strategy, coupled with a low number of epochs. The cosine annealing might not be adapted in this case because the learning rate is not kept high long enough for the model to capture sufficient information at the beginning of the training. As we can see in figure 3, the learning rate is around 0.01 for the first 50 epochs, but is then annealed for the rest of the training to a smaller value which does not encourage learning.

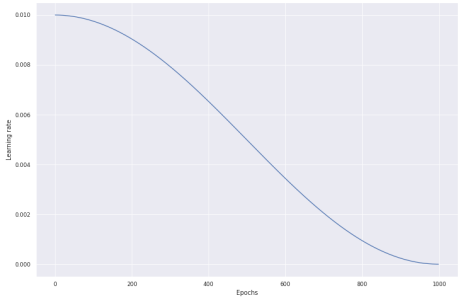


**Figure 3:** Cosine annealing for 1000 epochs

To address this issue, we decided to use a non constant division annealing strategy instead. The learning rate is first fixed to 5e-3, then divided by 10 at 300 epochs, set to 1e-4 at 600 epochs, and to 5e-5 at 1200. The MobileNet is trained for a total 1500 epochs. With less training epochs than previously, it achieved a better validation policy categorical accuracy of **43.3%** and a validation value MSE of **0.065**. By changing the annealing strategy, we were able to obtain a better result.

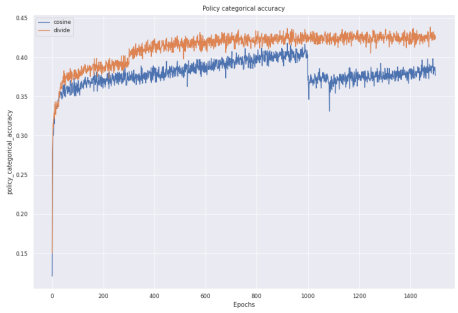


**Figure 4:** MobileNetV3, Cosine with restart at 1000 epochs (blue), Non constant division annealing (orange), Policy categorical accuracy.

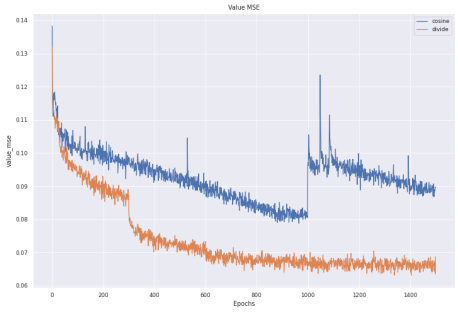


**Figure 5:** MobileNetV3, Cosine with restart at 1000 epochs (blue), Non constant division annealing (orange), Value MSE.

## 3.3 Third model: MixNet + SE + Swish

Inspired by [10], we investigated a MobileNetV3 backbone network on which depthwise convolutions were replaced by mixed convolutions. In order to limit parameter volume, we used atrous convolution operations in lieu of increasing kernel size. 5x5 kernels were thus replaced by 3x3 kernels with a dilation factor of 2, and 7x7 kernels by 3x3 kernels with a dilation factor of 3. This *atrous mixed convolution* operator was then plugged into the bottleneck inverted residual block containing the squeeze-excitation block, effectively creating the structure of our network.

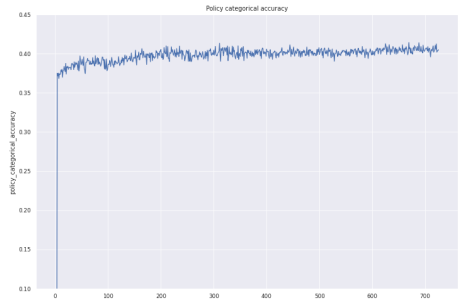


**Figure 6:** MobileNetV3 + MixConv, Policy categorical accuracy.

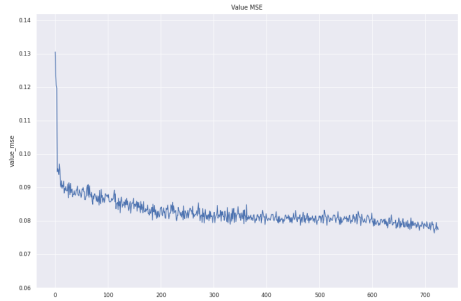


**Figure 7:** MobileNetV3 + MixConv, Value MSE.

As one is able to see in Figures 6 and 7 , MobileNetV3 provides good results compared to the base ResNet, albeit slightly worse than other architectures. We obtained a validation policy categorical accuracy of **41.2%** and a validation value MSE of **0.078**.

### 3.4 Fourth model: MobileNet + CBAM + Swish

As mentioned in Section 2.2.3, the squeeze-excitation block can be regarded as a channel attention module, and thus enables the network to learn "what" to focus its attention on. However, it misses attention along the spatial dimension, i.e. along the height and width dimensions, which can be as important to consider as the channel dimension in the game of Go. For instance, we would want to encourage the network to shift its attention on the part of the board with a concentration of pawns. This motivates us to investigate on other attention modules bringing spatial dimension to the table like CBAM (Section 2.3).

Our fourth model uses a MobileNetV3 architecture on which SE blocks were replaced by CBAM blocks. We adapted the code found here https://github.com/kobiso/CBAM-tensorflow/blob/master/attention_module.py for Tensorflow 2. The MobileNet-CBAM is composed of 24 bottleneck blocks, 512 filters and 128 trunks. It was trained with a batch size of 64, and Adam optimizer with a starting learning rate of 1e-3 for 500 epochs, then 5e-4 until 800 epochs, and finally 1e-4 until 900 epochs.

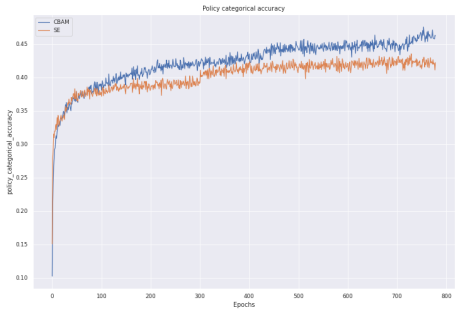


**Figure 8:** MobileNet-CBAM (blue), MobileNetV3 (orange), Policy categorical accuracy.

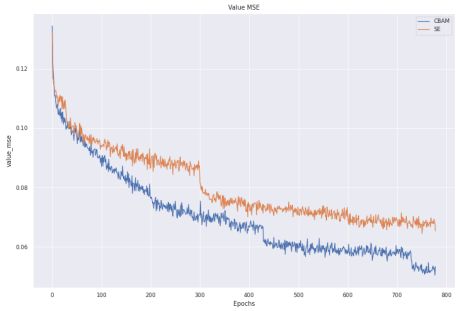


**Figure 9:** MobileNet-CBAM (blue), MobileNetV3 (orange), Value MSE.

MobileNet-CBAM provides the best result up until now with a validation policy categorical accuracy of **47.7%** and a validation value MSE of **0.051**, beating the MobileNet-SE with less training time. These results show the good impact of using CBAM as attention module rather than SE. The network looks promising: there is still room for improvement since we can set the learning rate to a lower value, and thus achieving better results. Unfortunately, we did not have the time to train for a higher number of epochs.

### 3.5 Fifth model: EfficientNet

Finally, in a bid to evaluate the current state-of-the-art convolutional neural network for image recognition on the task of computer Go, we implemented an EfficientNet-B4 with $\alpha = 1.8,\ \beta = 1.4$. However, the performance of the EfficientNet after several hundred epochs was not up to par with some of our best models. We thus decided to discard explorating this architecture further.

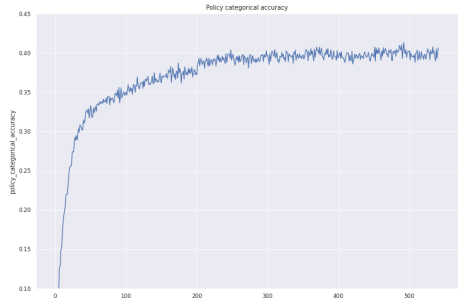


**Figure 10:** EfficientNet-B4, Policy categorical accuracy.

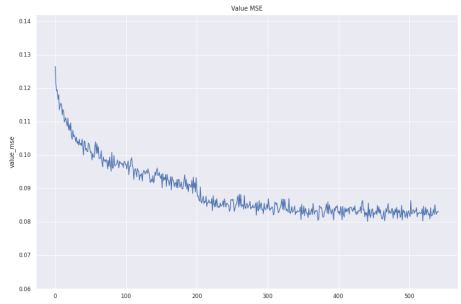


**Figure 11:** EfficientNet-B4, Value MSE.

## 4 Conclusion

During the course of this project, we investigated several state-of-the-art architectures for computer Go. After evaluating five different architectures, we report that our best performing model is the MobileNetV3 to which we incorporated the CBAM block. To the best of our knowledge, the approach of spatial attention has not yet been explored for the game of Go. We are proud to present results of **47.7%** policy accuracy and **0.051** value MSE. Even though our training efforts were limited by time constraints as well as computational limitations, we showed that this novel architecture is viable and may be worth exploring for future research. This project helped us gain knowledge on the current state-of-the-art deep convolutional neural network models.

## References


[1] Silver D. et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* (2016).

[2] He K. et al. "Deep Residual Learning for Image Recognition". In: *CoRR* (2015).

[3] Cazenave T. "Improved Policy Networks for Computer Go". In: *ACG 2017* (2017).

[4] Howard A. et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *CoRR* (2017).

[5] Sandler M. et al. "Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation". In: *CoRR* abs/1801.04381 (2018).

[6] Howard A. et al. "Searching for MobileNetV3". In: *CoRR* (2019).

[7] Hu J., Shen L., and Sun G. "Squeeze-and-Excitation Networks". In: *CoRR* (2017).

[8] Woo S. et al. "CBAM: Convolutional Block Attention Module". In: *CoRR* (2018).

[9] Tan M and Le Q. V. "MixConv: Mixed Depthwise Convolutional Kernels". In: *CoRR* (2019).

[10] Cazenave T., Sentuc T., and Videau M. "Cosine Annealing, Mixnet and Swish Activation for Computer Go". In: *Advances in Computer Games* (2021).

[11] Tan M and Le Q. V. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *CoRR* (2019).

[12] Cazenave T. "Mobile Networks for Computer Go". In: *CoRR* (2020).

[13] Cazenave T. "Improving Model and Search for Computer Go". In: *CoRR* (2021).