Report deep Learning project - Master IASD 2020-2021

Thomas Legand & Antonio Rausell

August 2021

## 

## **Table de matières:**

[**Table de matières:**](#_cqrecnm395sn) **1**

[**1. Objective**](#_xbwr09q3dmix) **2**

[**2. Concepts clés associés aux réseaux résiduels, aux MobileNets et aux Shufflenets**](#_sf2icfbkokpb) **2**

[**3. Moyens de calcul utilisés**](#_4nnvinna5c4n) **4**

[**4. Implémentation du réseau**](#_l35rx1h7s5t8) **4**

[**5. Rappel des résultats des 3 tournois**](#_5zralue5tn6q) **5**

[**6. Optimisation du MobileNetV2 et de ses paramètres**](#_5biv8ok8tcay) **5**

[6.1. Premier tournoi](#_2kpi278019sn) 5

[6.2. Deuxième tournoi](#_yotn8hlw0aj) 5

[6.3. Troisième tournoi](#_jphmmo73ydrg) 7

[**7. Combinaison en parallèle ou en série d’architectures MobileNet et ShuffleNet**](#_uw3pl7jcmxod) **8**

[**8. Conclusions et perspectives**](#_5iw6vwwfwoq7) **12**

[**9. Bibliographie**](#_3vmmsyi190rv) **12**

[**10. Notebooks annexes**](#_up4od9nbryuh) **13**

## **1. Objective**

Entraînement d'un réseau de Go avec moins de 1'000'000 de paramètres. Comme décrit dans <https://www.lamsade.dauphine.fr/~cazenave/DeepLearningProject.html> :

*The goal is to train a network for playing the game of Go. In order to be fair about training ressources the number of parameters for the networks you submit must be lower than 1 000 000. The data used for training comes from the Katago Go program self played games. There are 1 000 000 different games in total in the training set. The input data is composed of 21 19x19 planes (color to play, ladders, current state on two planes, two previous states on four planes). The output targets are the policy (a vector of size 361 with 1.0 for the move played, 0.0 for the other moves), and the value (1.0 if White won, 0.0 if Black won).*

*The project [...] uses Tensorflow 2.0 and Keras for the network. If you want to use dynamic batches of examples you should also install Pybind11. An example of a residual network with two heads is given in file golois.py and saved in file test.h5. The networks you design and train should also have the same policy and value heads and be saved in h5 format.*

## **2. Concepts clés associés aux réseaux résiduels, aux MobileNets et aux ShuffleNets**

Pour réaliser ce projet, nous nous sommes principalement appuyés sur l'exploration des architectures basées sur MobileNet et ShuffleNet. Nous citons ici certains des concepts clés derrière les réseaux résiduels ainsi que derrière l'efficacité de calcul atteinte par les petits réseaux pour les dispositifs portables sans perte de précision :

Citations tirées de *Deep Learning with Python, François Chollet (1st Ed, Manning, 2017):*

*A* ***residual connection*** *consists of making the output of an earlier layer available as input to a later layer, effectively creating a shortcut in a sequential network. Rather than being concatenated to the later activation, the earlier output is summed with the later activation, which assumes that both activations are the same size. If they’re different sizes, you can use a linear transformation to reshape the earlier activation into the target shape (for example, a Dense layer without an activation or, for convolutional feature maps, a 1 × 1 convolution without an activation). [...]* ***Residual connections*** *tackle two common problems in deep-learning: vanishing gradients and representational bottlenecks. In general, adding residual connections to any model that has more than 10 layers is likely to be beneficial.*

***Representational bottlenecks:*** *in a Sequential model, each successive representation layer is built on top of the previous one, which means it only has access to information contained in the activation of the previous layer. If one layer is too small (for example, it has features that are too low-dimensional), then the model will be constrained by how much information can be crammed into the activations of this layer. [...] [Across the different layers], any loss of information is permanent. Residual connections, by reinjecting earlier information downstream, partially solve this issue for deep-learning models.*

***Vanishing gradients in deep learning:*** *Backpropagation, works by propagating a feedback signal from the output loss down to earlier layers. If this feed-back signal has to be propagated through a deep stack of layers, the signal may become tenuous or even be lost entirely, rendering the network untrainable. This issue is known as vanishing gradients. [..] Residual connections introduce a purely linear information carry track parallel to the main layer stack, thus helping to propagate gradients through arbitrarily deep stacks of layers.*

Citations tirées de *Mobile Networks For Computer Go* (Cazenave 2020, <https://arxiv.org/abs/2008.10080>)

***Residual Networks:*** *The principle of residual networks is to add the input of a residual block to its output. A residual block is composed of two convolutional layers with ReLU activations and batch normalization.*

***MobileNetsv2:*** *The principle of MobileNetV2 is to have blocks as in residual networks where the input of a block is added to its output. But instead of usual convolutional layers in the block they use depthwise convolutions. Moreover the number of channels at the input and the output of the blocks (in the trunk) is much smaller than the number of channels for the depthwise convolutions in the block. In order to efficiently pass from a small number of channels in the trunk to a greater number in the block, usual convolutions with cheap 1x1 filters are used at the entry of the block and at its output.*

Citations tirées de *MobileNetV2: Inverted Residuals and Linear Bottlenecks, Sandler et al 2019,* <https://arxiv.org/pdf/1801.04381.pdf>

***MobileNetsv2*** *use* ***depthwise separable convolutions****: Depthwise Separable Convolutions are a key building block for many efficient neural network architectures [...]. The basic idea is to replace a full convolutional operator with a factorized version that splits convolution into two separate layers. The first layer is called a* ***depthwise convolution****, it performs lightweight filtering by apply-ing a single convolutional filter per input channel. The second layer is a 1 × 1 convolution, called a* ***pointwise convolution****, which is responsible for building new features through computing linear combinations of the in- put channels.*

Citations tirées de *Small But Powerful Convolutional Networks* <https://towardsdatascience.com/3-small-but-powerful-convolutional-networks-27ef86faa42d>

*A depthwise separable convolution is made of two operations: a* ***depthwise convolution*** *and a* ***pointwise convolution****. [...] The computational cost of the depthwise separable convolution is the sum of the costs of the depthwise and pointwise operations. Compared to a normal convolution it offers a computation reduction of 1/N + 1/Dk², with N the number of output channels, and Dk the kernel size. With a kernel size of 3x3, it results in 8 times less operations!*

Citations tirées de *ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices (Zhang et al 2017,* <https://arxiv.org/abs/1707.01083>) et du blog associé :

<https://medium.com/syncedreview/shufflenet-an-extremely-efficient-convolutional-neural-network-for-mobile-devices-72c6f5b01651>

***ShuffleNet****: CNN architecture that utilizes two new operations, pointwise group convolution and channel shuffle, to greatly reduce computation cost while maintaining accuracy. [...] The main constituent architecture of ShuffleNet involves a novel channel shuffle operation used to help information flow across feature channels*

Citations tirées deSmall But Powerful Convolutional Networks <https://towardsdatascience.com/3-small-but-powerful-convolutional-networks-27ef86faa42d>

*A* ***ShuffleNet*** *is composed of group convolutions and channel shuffles. [...] A group convolution is simply several convolutions, each taking a portion of the input channels. [...] It greatly diminishes the computational cost. [...] A channel shuffle then randomly mixes the output channels of the group convolution.*

## **3. Moyens de calcul utilisés**

Nous avons utilisé les moyens de calculs de deux environnements pour réaliser ce projet Golois : (i) Colab Pro et (ii) Un serveur local avec un processor Intel(R) Xeon(R) Gold 6140 CPU @ 2.30GHz , 640Gb RAM et 2 GPU’s NVIDIA GeForce RTX 2080 Ti Rev. A, 11 GB, installé avec CentOS Linux 7 (Core).

## **4. Implémentation du réseau**

Nous avons implémentés quelques réseaux naïfs pour démarrer mais ils ont été infructueux :

* Réseaux convolutionnels multi-couches simples vus sur Kaggle
* Réseau résiduel à 7 blocs inspiré de “*Cross-width convolutions by Jeffrey Barratt*”

Ils sont visibles dans le Notebook ***legrand-raussel-premiers-essais.ipynb.***

* Residual network with two heads (policy and value):

C’est la lecture de *Mobile Networks For Computer Go* (Cazenave 2020, <https://arxiv.org/abs/2008.10080>)et l’implémentation du *MobileNetV2* comme décrite dans l’appendix de l’article qui permet d’obtenir de premiers bons résultats (sous la contrainte de 106 paramètres) et de lancer l’exploration des paramètres. Nous fournissons tous les détails dans les sections en aval.

## **5. Rappel des résultats des 3 tournois**

1. 19 wins, 26 games played, winrate = 0.731 , sigma = 0.087
2. 23 wins, 32 games played, winrate = 0.719 , sigma = 0.079
3. 20 wins, 34 games played, winrate = 0.588 , sigma = 0.084

## **6. Optimisation du MobileNetV2 et de ses paramètres**

### **6.1. Premier tournoi**

Pour le **1er tourno**i, nous avons mené plusieurs phases exploratoires du réseau *mobile.small.conv.avg.bin* : sur la structure des blocs puis, une fois l’architecture figée, sur le *learning rate* et les *weights* sur les objectifs *policy* et *value*.

* Choix définitifs sur la structure :
  + 32 blocs de bottleneck\_block
  + Structure fine à 64 filtres, expansion à 200 filtres
* 5 phases de 100 epochs ont été réalisées avec des variation de :
  + N = 100000, batch size = 128
  + Décroissance du learning rate :
    - step\_decay : lrate = initial\_lrate \* math.pow(drop, math.floor((1+epoch)/epochs\_drop))
    - lr\_scheduler : inspiré des points d’inflexion de l’*accuracy* à 100 et 150 epochs de l’article de recherche
  + Les valeurs (1,2) et (1,4) sur le couple de *weights (policy, value)* n’ont pas permis d’améliorer nos valeurs.

### **6.2. Deuxième tournoi**

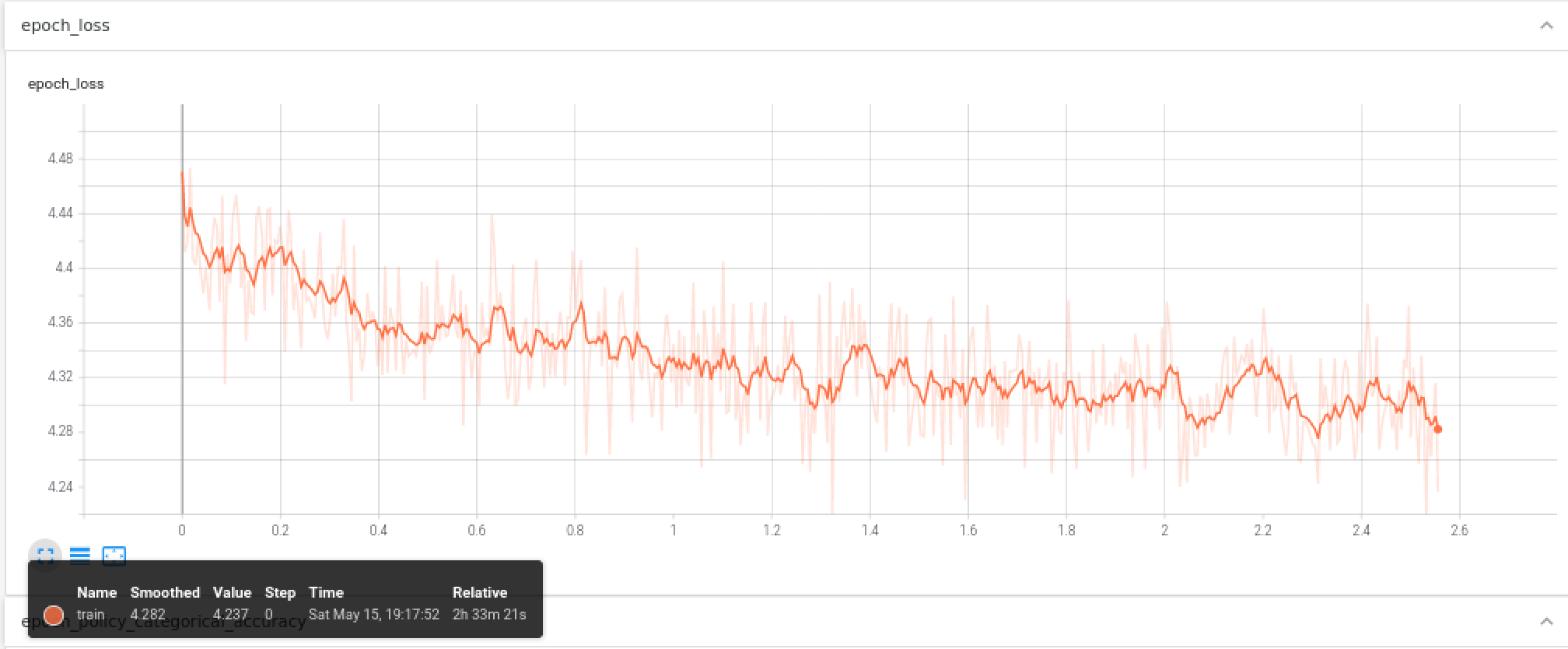
Pour le **2ème tourno**i, nous avons poursuivi l’apprentissage avec une architecture similaire (33 blocs *bottleneck\_block\_mobilenet* avec 198 filtres) sur plus de 500 epochs et utilisé ReduceLROnPlateau plutôt que nos implémentations manuelles. Le code complet utilisé dans ce deuxième phase est visible dans le notebook attaché **importGolois\_2ndTrain.ipynb**

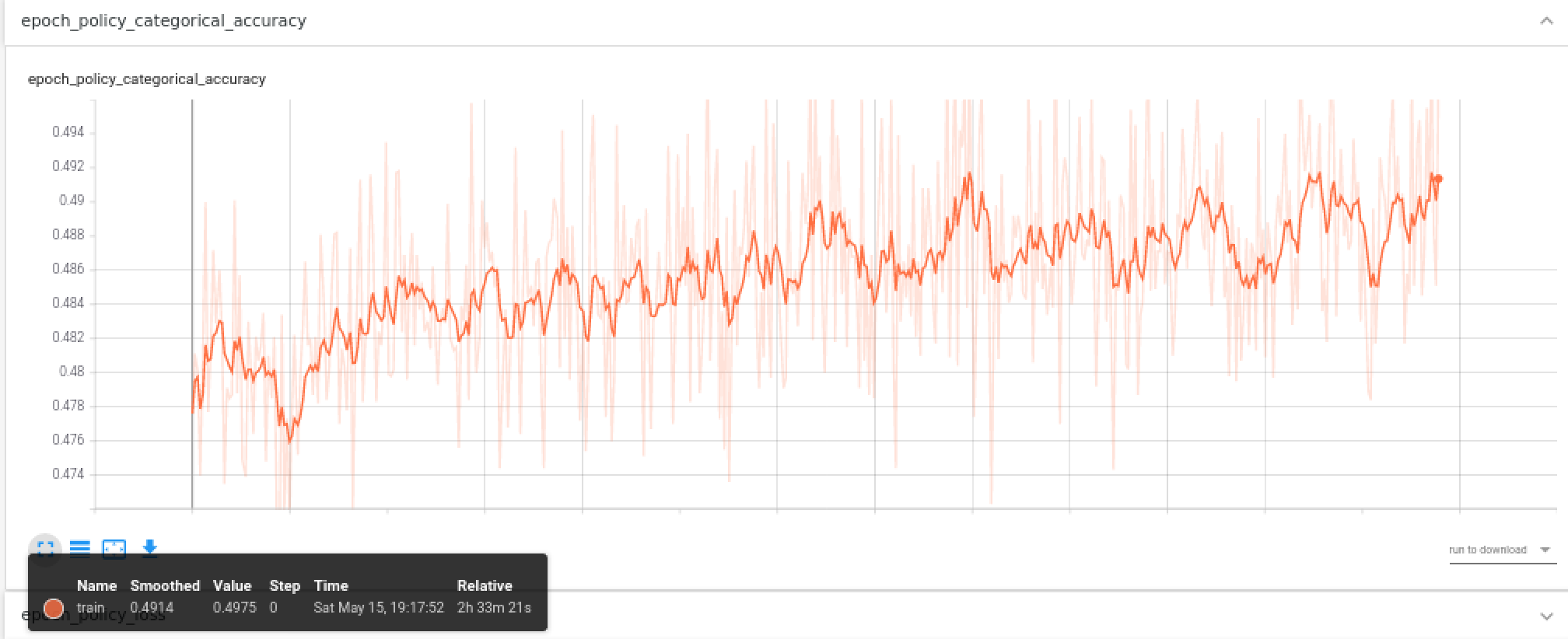
Nous reproduisons ici la configuration de la callback\_list utilisée, avec les paramètres utilisés pour la fonction ReduceLROnPlateau :



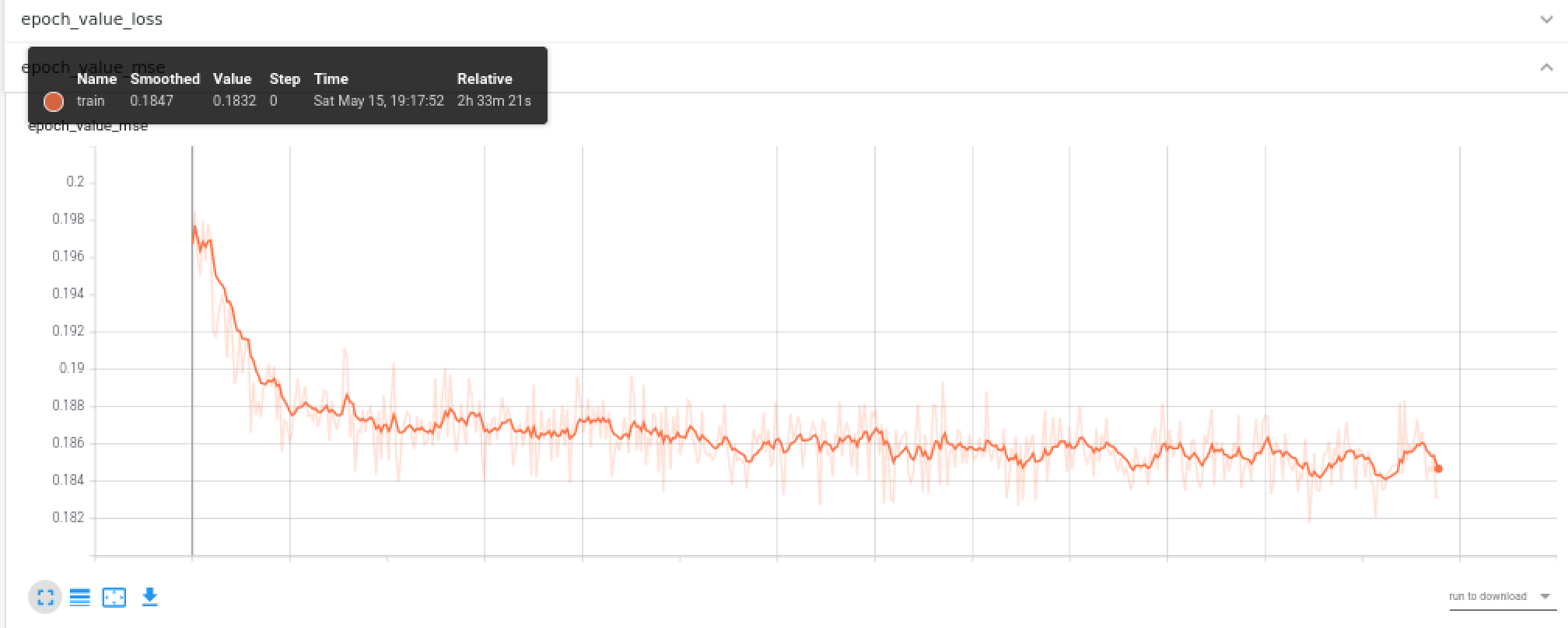
On a donc établi : N=10000; batch=64 ; Policy\_weight= 2 ; Value\_weight=1 ; optimizer SGD avec learning rate initial = 0.005 et momentum=0.9 et keras.callbacks.ReduceLROnPlateau( monitor='loss', factor = 0.1, patience = 10), epoch=500.

Avec cette configuration, le modèle retenu à obtenu une accuracy = 0.4975 et mse = 0.1832. Nous reproduisons ci-dessous l’évolution de la loss, accuracy et mse dans ce deuxième entraînement telle que representé avec l’outil **TensorBoard**. On observe comment le training semble avoir saturé.

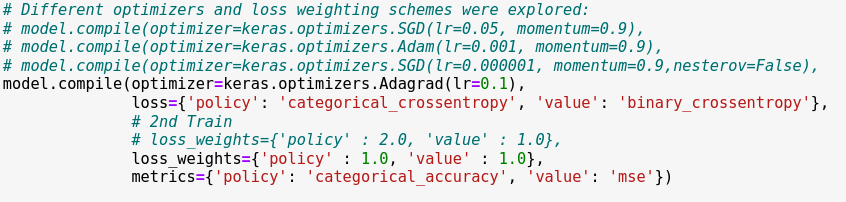




value\_mse:



Par ailleurs, nous avons exploré différents types d’optimisers (SGD, Adam et Adagrad, avec différents learning rates et momentum, y compris Nesterov), ainsi que différents poids pour la policy et le value comme montre le *snippet* suivant:



Par contre, nous n’avons pas observé une amélioration importante entre les différents *optimisers* évalués.

### **6.3. Troisième tournoi**

Pour le 3ème tournoi, nous avons utilisé le framework *HyperOpt* (et une courte formation réalisée par Databricks) pour essayer de faire mieux que notre approche manuelle de grille de paramètres :

* il a fallu agréger les deux objectifs pour construire la métrique :

obj\_metric = **3.0** \* (history.history["**value\_loss**"][-1]) **-** history.history["**policy\_loss**"][-1]

* puis définir un espace d’exploration sur :
  + un taux d’activation *alpha leakly* dans les blocs résiduels
  + un taux de régularisation variable dans ces blocs

space = {

"L2\_regul": hp.uniform("l2\_rate", 0.0, 0.0005),

"leakly\_alpha": hp.uniform("leakly\_alpha", 0.0, 0.2)

"activation": hp.uniform("alpha\_rate", 0.0, 0.1)

}

* définition de stratégies d’exploration : un seul choix exploré *Tree-structured Parzen Estimator Approach* (TPE) après revue rapide de [Algorithm For Hyper-Param Optimization](https://papers.nips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf)

with mlflow.start\_run():

best\_hyperparam = fmin(fn=runNN,

space=space,

algo=**tpe.suggest**,

max\_evals=5,

trials=colab\_trials,

rstate=np\_rstate)

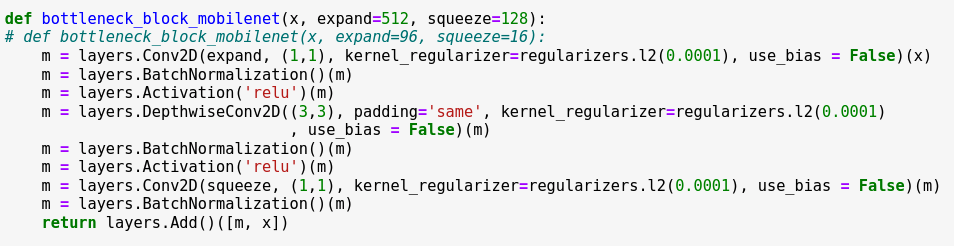
* Les résultats ont été décevants :
  + l’exploration était beaucoup trop longue pour être large
  + les meilleurs paramètres trouvés (les deux types de L2, activations) ont été appliqués sur 500 epochs : la progression (policy et value) était continue mais beaucoup trop lente comparée aux paramètres trouvés manuellement dans les 2 premières phases du tournoi
  + l’objectif pondéré est inadapté et délivre un mauvais compromis : du code plus complexe avec HyperOpt serait à explorer pour définir 2 objectifs et adapter la stratégie d’exploration TPE

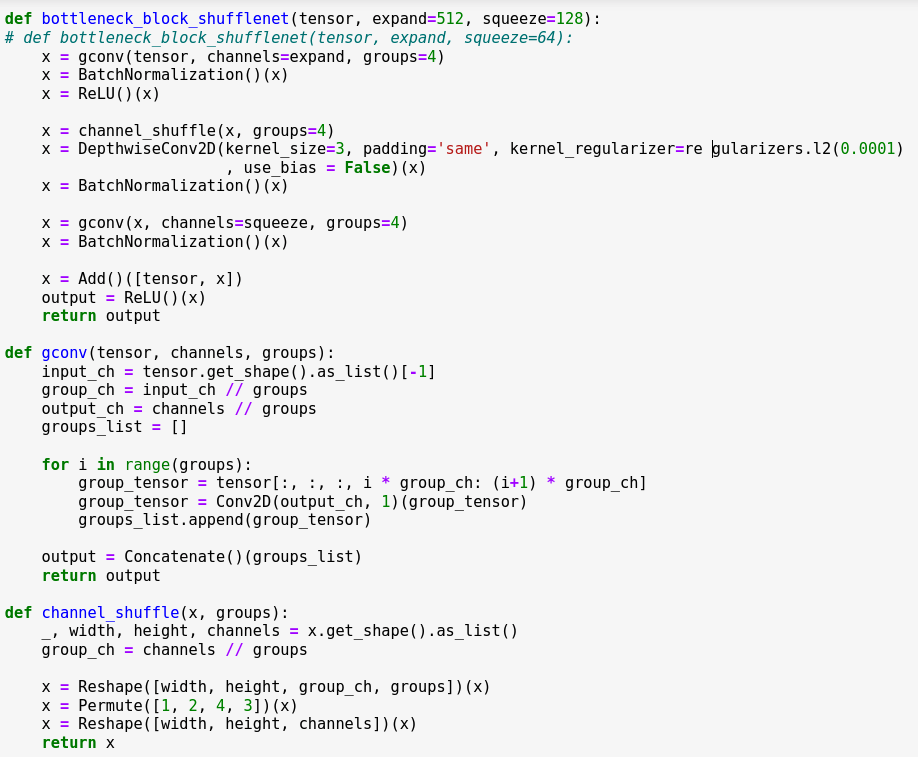
Le troisième notebook nommé **importGolois\_3rdTrain.ipynb** rassemble ces travaux.

## **7. Combinaison en parallèle ou en série d’architectures MobileNet et ShuffleNet**

D'un point de vue expérimental, nous avons également mis en œuvre une architecture combinant une architecture MobileNet et une architecture ShuffleNet. Une telle combinaison peut être réalisée avec les deux types en parallèle ou en série (par exemple, en appliquant le ShuffleNet après les blocs de construction MobileNet). La configuration parallèle est facilement réalisable avec l'API fonctionnelle de KERAS, qui permet de gérer des topologies non linéaires, traitées comme des graphes acycliques dirigés (DAG) de couches. Le notebook correspondant est fourni en fichier joint **importGolois\_MobileNet\_ShuffleNet\_combined.ipynb**

Nous reproduisson ici les snippets de code où on montre l'implémentation du bloc MobileNet ainsi que du bloc ShuffleNet:





À partir de ces fonctions, on peut définir une combinaison en parallèle comme suit:



==================================

Total params: 971,661

Trainable params: 925,613

Non-trainable params: 46,048

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Autrement, on peut aussi les combiner **en série** comme se montre dans le snippet suivant::



=============================

Total params: 968,397

Trainable params: 922,349

Non-trainable params: 46,048

Malgré la beauté des idées précédentes, les résultats obtenus étaient loin d'être brillants. Ainsi, avec des paramètres équivalents à ceux utilisés dans le deuxième entraînement (voir ci-dessus) et en utilisant cette fois 50 épochs, ni la configuration parallèle ni la configuration série n'ont réussi à atteindre des chiffres d'entraînement décents :

Résultats ***en parallèle*** :

* loss: 6.8077
* policy\_loss: 5.8889
* value\_loss: 0.6644
* policy\_categorical\_accuracy: 0.0015
* value\_mse: 0.2366

Résultats **en série** :

* loss: 6.8341
* policy\_loss: 5.8889
* value\_loss: 0.6931
* policy\_categorical\_accuracy: 0.0019
* value\_mse: 0.2500

À la lumière des faibles performances obtenues à 50 épochs, nous avons abandonné l'exploration de ces paramètres et sommes revenus à l'architecture originale utilisée dans la deuxième formation ci-dessus.

## 8. Conclusions et perspectives

Tout au long de ce projet, nous avons exploré différentes architectures de réseaux résiduels (MobileNet V2 et ShuffleNet) appliquées à l'apprentissage de jeux GO à deux têtes (politique et valeur). Nous avons exploré différents paramètres concernant à la fois l'architecture (par exemple, le nombre de blocs, les filtres, la taille des lots) et le processus d'apprentissage (type d'optimiseur, schémas de pondération pour la valeur et la perte, taux d'apprentissage, adaptateur de taux d'apprentissage). D'un point de vue plus avancé, nous avons également implémenté l'approche d'optimisation automatisée des hyperparamètres HyperOpt ainsi que des architectures mixtes combinant des blocs MobileNet et ShuffleNet en parallèle et en série. Compte tenu d'un paramétrage fixe à deux têtes (politique et valeur) et avec la contrainte de 1'000'000 de paramètres, nous avons eu l'impression d'avoir atteint la saturation du processus d'apprentissage à travers les différentes options évaluées. Ainsi, la plupart des alternatives évaluées pour optimiser le modèle ont soit réduit de manière significative les performances, soit les ont augmentées de manière différentielle et négligeable. De plus, en l'absence d'un ensemble de données de test, nous ne pouvions pas savoir si les gains mineurs en termes de précision et de perte représentaient de véritables améliorations du modèle ou simplement un surajustement. Une percée substantielle nécessiterait probablement plus de créativité dans l'architecture CNN et c'est probablement un sujet actif de recherche académique.

## **9. Bibliographie**

Deep Learning with Python, François Chollet (1st Ed, Manning, 2017):

MobileNetV2: Inverted Residuals and Linear Bottlenecks, Sandler et al 2019,

<https://arxiv.org/abs/1801.0438>

Mobile Networks For Computer Go, Cazenave 2020, <https://arxiv.org/abs/2008.10080>

ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. Zhang et al 2017. <https://arxiv.org/abs/1707.01083>.

Blog “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices”: https://medium.com/syncedreview/shufflenet-an-extremely-efficient-convolutional-neural-network-for-mobile-devices-72c6f5b01651

Blog “3 Small But Powerful Convolutional Networks”

https://towardsdatascience.com/3-small-but-powerful-convolutional-networks-27ef86faa42d

Helpful graphical representations to understand 1x1 point-wise convolutions:

Figure 7 in <http://www.pabloruizruiz10.com/resources/CNNs/Convolution_Pooling.pdf>

Figure 3 in <http://www.pabloruizruiz10.com/resources/CNNs/1x1Convolutions.pdf>

## **10. Notebooks annexes**

* *legrand-raussel-premiers-essais.ipynb.*
* importGolois\_2ndTrain.ipynb
* importGolois\_MobileNet\_ShuffleNet\_combined.ipynb
* importGolois\_3rdTrain.ipynb