Deep Learning project

Computer Go

February 24, 2021

Tiago Fleury
Hugo Da Costa

Supervisor:

Tristan Cazenave

M2 Intelligence Artificielle, Systèmes, Données
Mathématiques et Informatique de la Décision et des Organisations
University of Paris Dauphine - PSL

Table des matières

1. Introduction

Au sein de ce document, nous vous présentons notre travail et nos résultats du projet de deep learning “Computer Go”. Ce projet a pour but de concevoir une architecture de réseau de neurones ainsi que d’optimiser son entraînement afin de challenger les résultats obtenus par l’état de l’art de la discipline, ainsi que les différents réseaux du Master IASD lors d’un Tournoi.

L'entraînement de ce réseau consiste en l’optimisation de différentes métriques telles que la Policy Accuracy et la Value MSE. Pour cela, nous disposons de données d’entrée chacunes représentées par 21 plans de taille 19*19. Ces données représentent des plateaux de jeu de Go et sont associées aux coups et valeurs donnés par KataGo, référence en termes de coups et d’évaluation du jeu de Go.

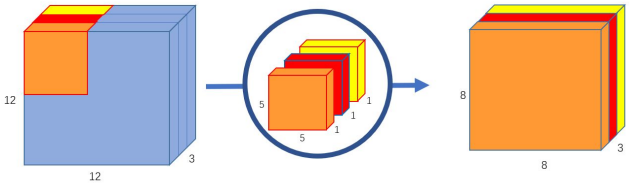
Tout en suivant la contrainte de ne pas excéder 1 millions de paramètres, notre réseau s’inspire d’une architecture récente de réseau performant en computer vision. Cette architecture nommée Mobile Network (MobileNetV2, Google, 2018) qui se veut très légère a été étudiée dans le cadre de l’apprentissage du jeu de Go (avec contrainte sur le nombre de paramètres) par notre professeur Tristan Cazenave dans son papier *Mobile Networks for Computer Go* avec certaines variations dans l’organisation et la composition des couches de convolution.

2. Architecture

Les Mobile Networks sont des variantes des **Residuals networks** (dont l’application à la problématique du jeu de Go a offert un véritable tournant de performance), dans lesquels certaines couches de convolution sont remplacées par des couches de **Depth wise convolution**.

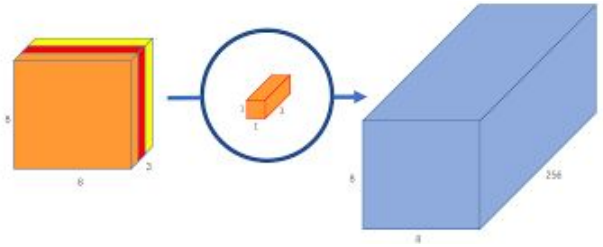
La particularité de ces couches de depthwise convolution s’étudie en 2 étapes (ici on souhaite produire 256 filtres par exemple) :

- Le passage d’un kernel de taille *k * k * 1* sur chaque channel d’entrée afin de produire *c* pseudo-filtres (avec *c* le nombre de channels).



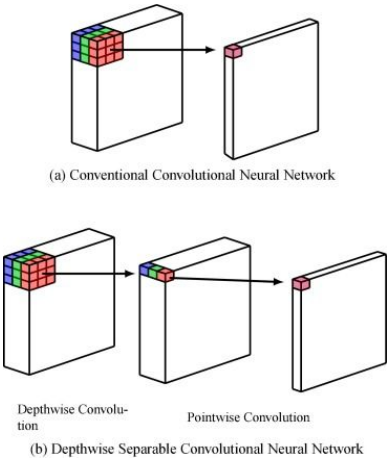
Source : Towards Data science : A basic intro to depth wise convolution

- Une convolution classique de kernel_size = *1 * 1 * c* afin de produire les 256 filtres voulus.



Source : Towards Data science : A basic intro to depth wise convolution

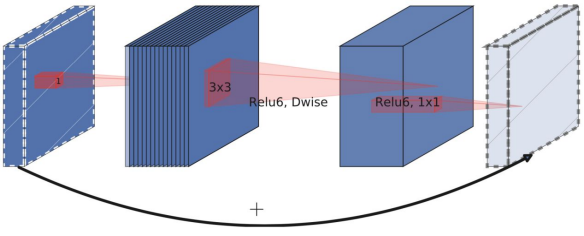
Cette technique en deux étapes permet de réduire considérablement le nombre d'opérations effectuées ainsi que les paramètres du réseau. En comparaison des couches de convolutions classiques, il s'agit dans notre cas de près de 10 fois moins de paramètres. En pratique on n'observe qu'une légère baisse de performances par rapport aux convolutions classiques.



Source : sciencedirect.com

L'apport de ces convolutions depth wise est donc la réduction conséquente du nombre de paramètres du réseau tout en gardant l'assurance d'avoir la profondeur et la largeur suffisante pour apprendre à représenter l'information le mieux possible et donc avoir un entraînement plus complet au fil des epochs.

De plus, ces couches de depth wise convolution sont associées à des variations de la taille des channels : **blocks en Bottle neck**.

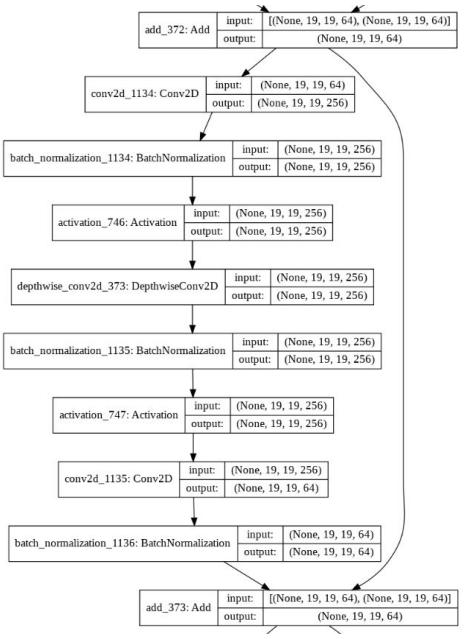


Source : "MobileNets: Efficient Convolutional …"

L'architecture finale que nous proposons est alors :

- **26** blocks en bottleneck constitués de
  - Un tronc de **64 filtres**
  - DepthWise conv **4*64 = 256 filtres**
  - Batch Normalisation
  - Activation ReLu

- régularisation **L2** de **0.0001**
- Un résidu au dessus de chaque bloc
- 2 Heads : respectivement Policy et Value heads, chacune des couches de convolutions de kernel 1



Le tout pour un total de **976,805** paramètres.

Entraînement et résultats

Contexte

Afin d'entraîner les réseaux en un temps raisonnable, nous avons utilisé les GPU de Google Colab. L'allocation des GPUs par Google est aléatoire. Une astuce que nous avons utilisé pour essayer d'avoir le plus souvent possible un des meilleurs GPU (Nvidia P100 > Nvidia T4 > Nvidia K80) est la suivante :

- Réinitialiser l'environnement d'exécution et exécuter la commande

```
[1]  1 !nvidia-smi -L

GPU 0: Tesla T4 (UUID: GPU-30297e83-80fd-9354-3339-12897273eb98)
```

  Si le GPU ne convient pas , réessayer.

- Parfois quelques réinitialisations ont suffit à tomber sur un P100. Ce qui nous a permis d'entraîner environ 5 fois plus vite qu'avec un K80

Paramètres d'entraînement

Pour entraîner notre réseau, nous commencions toujours par un learning rate de 0.005. Ce learning rate ayant vocation à diminuer au cours de l'entraînement selon les différents plateaux atteints, nous avons finalement utilisé :

- lr = 0.005 pour les 300 premières epochs
- lr = 0.0005 pour les 200 suivantes

- lr = 0.00005 pour les 100 dernières

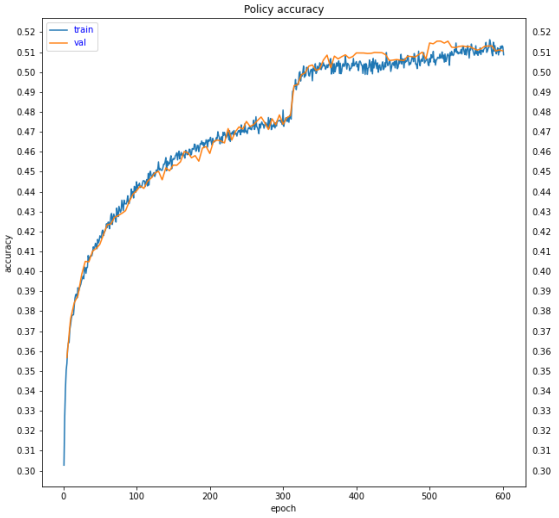
Nous avons donc entraîné le réseau sur 600 epoch. Une époque représente 100 000 parties vues dans toute la première partie de l'entraînement (300 premières epoch). Puis nous passons à des epoch de 70 000 pour des raisons que nous vous détaillerons plus bas.

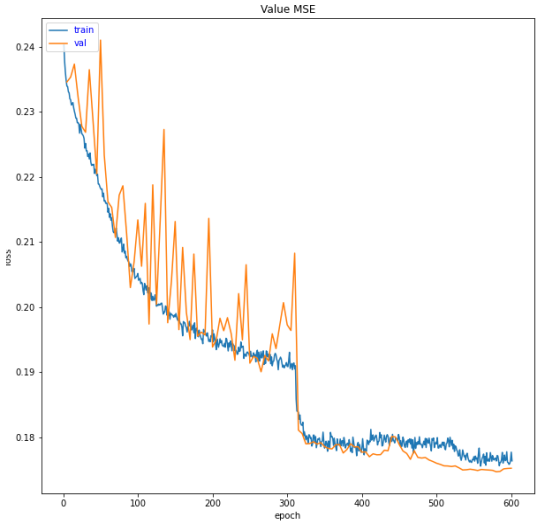
Résultats

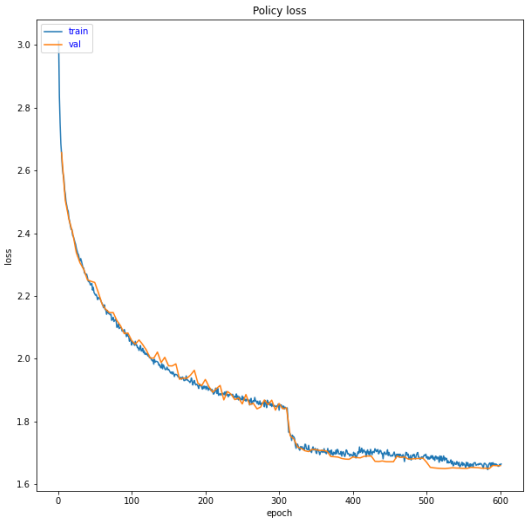
En validation, nous avons réussi à atteindre :

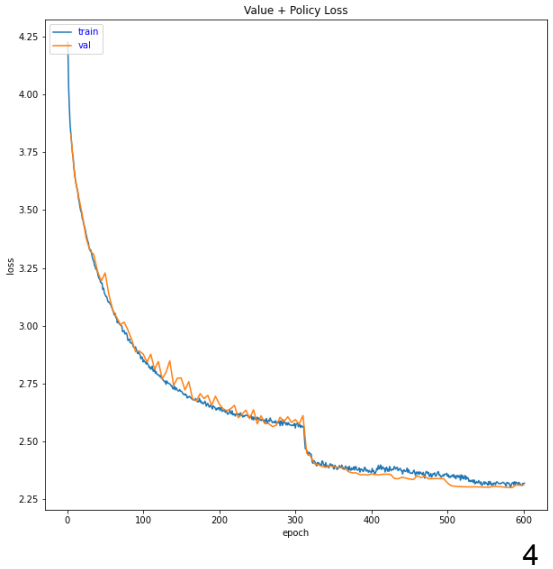
- **51.35%** de policy accuracy
- **0.1745** de value MSE
- **1.676** en policy loss
- **2.324** en loss

Ci-joint les courbes concernant ces métriques. Les performances en validation sont relevées toutes les 5 epoch.









Difficultés rencontrées et observations

Suite à des problèmes de colab intervenus vers le milieu de l'entraînement, nous avons dû terminer l'entraînement du réseau sur GCP. Pour des questions de coût, nous avons choisi une machine avec moins de RAM et réduit les epoch à 70 000 exemples.

Les points à retenir pour notre entraînement sont l'initialisation et l'optimizer.

Nous avons eu beaucoup de mal à faire "partir" nos architectures contenant des DepthWiseConv. Les problèmes rencontrés furent :

- La value head qui n'apprend pas.
- L'accuracy qui décroit jusqu'à converger vers 0.
- Des drops d'accuracy au milieu de l'entraînement (elle tombait à 0 pendant quelques epoch puis revenait à la normale).
- Un apprentissage globalement lent.
- Les changements de learning rate qui n'apportent pas l'impact espéré.

Tout d'abord, le passage de l'optimizer Adam à l'optimizer SGD a réglé le problème de l'accuracy qui tombait à 0. Ensuite, c'est en lançant de multiples initialisations et en ne conservant que les réseaux ayant eu le meilleur départ que nous avons réussi à obtenir trois modèles prometteurs dont les courbes laissaient espérer une bonne performance à long terme. Enfin, arrivés aux alentours de 48% d'accuracy, nous n'avons continué à entrainer que le meilleur des trois : celui que nous avons présenté au tournoi final et sur lequel nous n'avions pas encore eu besoin de diviser le learning rate. La première division de Learning Rate (0.005 -> 0.0005) aux alentours de 300 epoch a fait faire un bond à toutes les métriques (+3% en accuracy). La deuxième division (0.0005 -> 0.00005), aux 500 epoch, n'a cependant apporté qu'une très légère hausse. Cela nous mena donc une policy accuracy de près de 51.35%.

Conclusion

Au cours de nos travaux, nous avons découvert, approfondi et mis en place une architecture qui nous a apporté de bons résultats, notamment grâce à sa légèreté qui nous a permis d'avoir une profondeur suffisante pour entraîner de manière complète notre réseau, et de le perfectionner en utilisant un learning rate plus bas. Ce projet nous a permis d'entrer dans la problématique complexe du jeu de Go en découvrant et appliquant des architectures issues de l'état de l'art en deep learning.

Références

Cazenave, T. : Mobile networks for computer Go. IEEE Transactions on Games (2020)

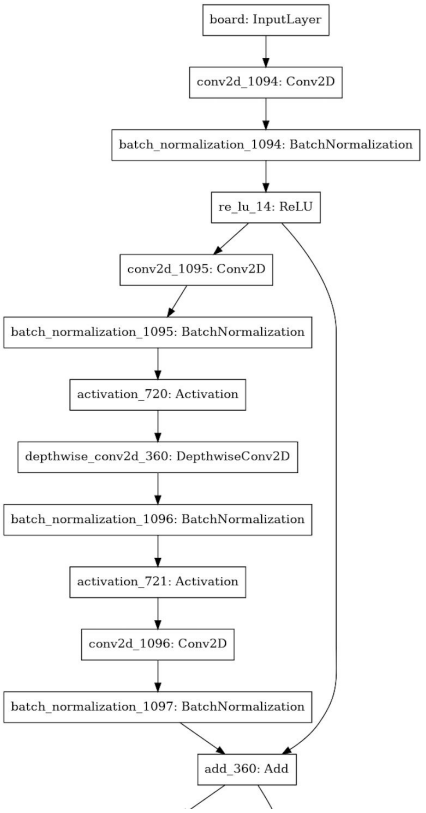
Cazenave, T. : Residual networks for computer Go. IEEE Transactions on Games 10(1), 107–110 (2018)

Andrew G. Howard, et al. : MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications (2017)

Cazenave, T., Chen, Y.C., Chen, G.W., Chen, S.Y., Chiu, X.D., Dehos, J., Elsa, M., Gong, Q., Hu, H., Khalidov, V., Cheng-Ling, L., Lin, H.I., Lin, Y.J., Martinet, X., Mella, V., Rapin, J., Roziere, B., Synnaeve, G., Teytaud, F., Teytaud, O., Ye, S.C., Ye, Y.J., Yen, S.J., Zagoruyko, S. : Polygames : Improved zero learning. ICGA Journal 42(4) (December 2020)

Google Inc. : MobileNetV2: Inverted Residuals and Linear Bottlenecks

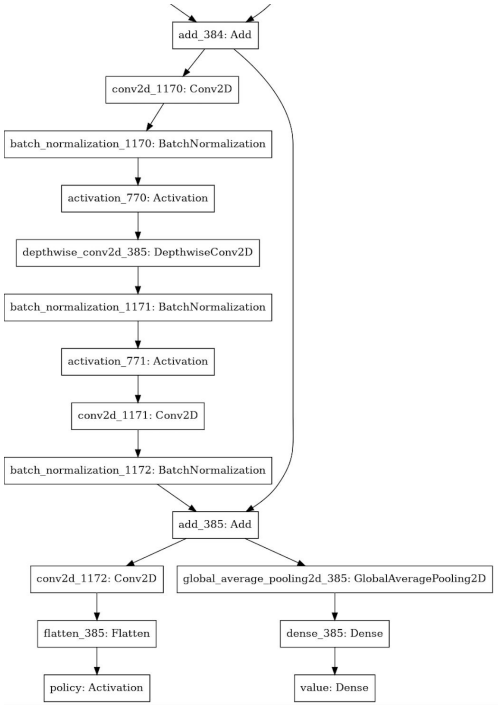
Annexe


board: InputLayer
conv2d_1094: Conv2D
batch_normalization_1094: BatchNormalization
re_lu_14: ReLU
conv2d_1095: Conv2D
batch_normalization_1095: BatchNormalization
activation_720: Activation
depthwise_conv2d_360: DepthwiseConv2D
batch_normalization_1096: BatchNormalization
activation_721: Activation
conv2d_1096: Conv2D
batch_normalization_1097: BatchNormalization
add_360: Add

...

x24

...


add_384: Add
conv2d_1170: Conv2D
batch_normalization_1170: BatchNormalization
activation_770: Activation
depthwise_conv2d_385: DepthwiseConv2D
batch_normalization_1171: BatchNormalization
activation_771: Activation
conv2d_1171: Conv2D
batch_normalization_1172: BatchNormalization
add_385: Add
conv2d_1172: Conv2D
global_average_pooling2d_385: GlobalAveragePooling2D
flatten_385: Flatten
dense_385: Dense
policy: Activation
value: Dense