Rapport sur le Projet de Deep Learning : Réseau Neuronal pour le Jeu de Go

Membres de l'Équipe :
- **Ismail BOUZKKA**
- **Anas FADIL**

Aperçu du Projet

Ce rapport détaille la conception et la mise en œuvre d'un réseau de neurones profonds pour jouer au jeu de Go. L'objectif est de créer un modèle capable de prédire efficacement les prochains coups ainsi que l'issue de la partie (victoire de Blanc ou de Noir) en utilisant un ensemble de données issu des parties auto-jouées par le programme Katago. La contrainte principale est de limiter le nombre de paramètres du réseau à moins de 100 000.

A travers ce rapport, nous allons présenter les approches pour lesquelles on a opté afin d'améliorer la performance de notre solution.

ResNet

Dans un premier temps, on a opté pour l'utilisation des réseaux résiduels (ResNets vu que c'était la première architecture vue lors du cours de Deep Learning).

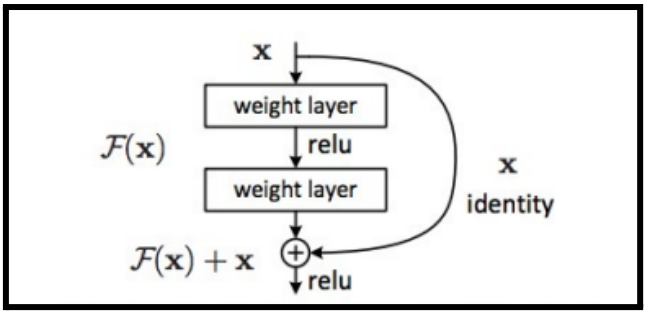


Figure 1: Bloc résiduel (ResNet)

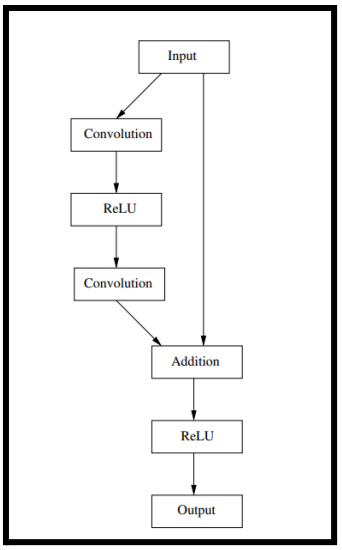


Figure 2: Couche résiduelle

Le principe des ResNets réside dans l'introduction de blocs résiduels qui permettent d'éviter le problème du Vanishing Gradient quand la profondeur du réseau devient importante.
En effet , chaque bloc apprend, tel qu'illustré dans la figure, une fonction résiduelle par rapport à l'entrée de la couche précédente. Cette approche se concrétise à travers l'incorporation d'une connexion de saut: cela consiste à ajouter l'entrée d'un bloc résiduel à sa sortie. Celui-ci se compose de deux couches convolutionnelles avec des activations ReLU (ainsi que des couches de batch normalisation).

Entrainement

L'approche suivie pour entraîner le réseau est la suivante:

- Un jeu de données avec une taille de 15000 .
- Un batch size fixé à 64.
- Un learning rate fixé à 0.005 .
- Un nombre d'époques fixé à 100.

Performance et résultat

Le premier entraînement a donné une accuracy de 38%. Cette performance a incité à explorer une deuxième architecture et une nouvelle méthodologie d'entraînement, détaillées dans la section suivante.

Convnext

Dans cette nouvelle approche, le choix a été fait sur un modèle basé sur un ConvNeXt considéré comme une meilleure alternative aux réseaux résiduels classiques, ce qui offrira une meilleure performance sur les données.

Architecture du bloc ConvNeXt V1

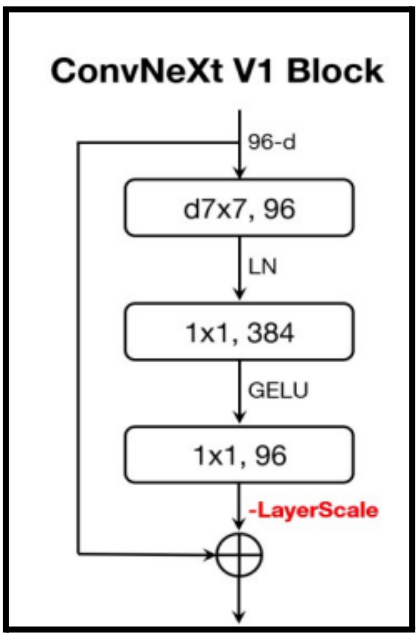


Figure 3: Bloc ConvNeXt V1

Un bloc convNeXt V1 est composé de :

- 7×7 DepthWise Convolution suivie d'une batch normalisation.
- 1×1 Convolution avec 4xfiltres suivie d'une GELU comme fonction d'activation.
- 1×1 Convolution avec filtres (Modification faite par nous).
- Ajout de la connexion résiduelle.

Modèle implémenté

Notre modèle, comprenant deux têtes (une pour la policy et une pour la value), totalisant 93939 paramètres, se compose de 9 blocs ConvNeXt V1.

Pour la policy, une convolution suivie d'un flattening sont effectuées. En revanche,pour la value, on a utilisé un Global Average Pooling afin de réduire la dimensionnalité et améliorer la robustesse du modèle

L'architecture simplifiée du modèle est représentée dans les annexes (où un seul bloc ConvNeXt est illustré au lieu de 9 pour des raisons de clarté) .
Lors de l'entraînement, on utilise la "Binary Crossentropy" comme loss pour la "value head" et la" Categorical Crossentropy loss" pour "la policy".
En ce qui concerne l'évaluation du modèle, les métriques utilisées sont les suivantes :

- La "categorical accuracy" pour la " policy".
- L'erreur quadratique moyenne (MSE) pour la "value".

Le batch size a été fixé 32 pour l'entraînement du modèle (plusieurs batch size ont été essayés : 64, 128). L'approche adoptée pour ajuster le learning rate est la suivante:
On a entrainé avec un learning rate initial de 0.005 pendant les 200 premières époques, ensuite on divisé sa valeur par 10 (0.0005) pour les 300 époques suivantes
Une régularisation L2 a été également appliquée avec un poids de 0.0001 afin de limiter le surapprentissage du modèle.
Finalement le modèle a été compilé une première fois avec l'optimiseur Adam, avant de basculer vers l'optimiseur AdamW lors du ré-entraînement du modèle (AdamW est une variante d'Adam qui inclut une décroissance des poids pour la régularisation, ce qui peut conduire à une meilleure généralisation du modèle).

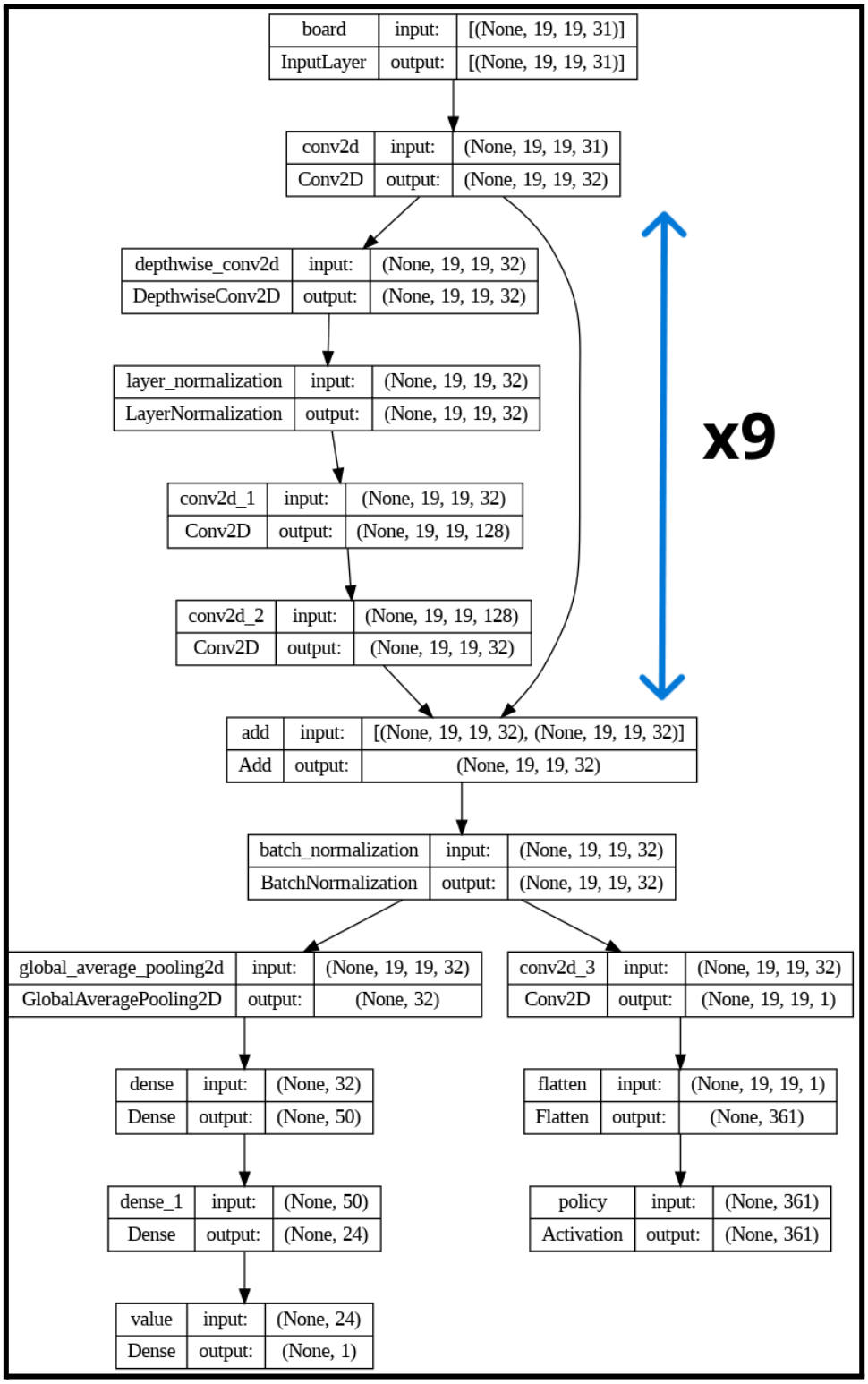
Conclusion

Après avoir soigneusement conçu et mis en œuvre notre approche, et après nos différentes expérimentations qui nous ont amené à pouvoir optimiser certains paramètres, nous avons obtenu une précision notable de 47,7% (Avec un nombre total d'époques égale à 1000).
Notre décision d'utiliser ConvNeXt comme alternative aux réseaux résiduels traditionnels a joué un rôle important. De plus, notre ajustement du learning rate, associé à l'application de la régularisation ont amélioré la capacité de généralisation du modèle et donc la réduction du surajustement (overfitting).
Le passage de l'optimiseur Adam à AdamW dans notre cas a également amélioré la performance du modèle, offrant une meilleure régularisation et convergence lors de l'entraînement.

En conclusion, notre approche, dans laquelle des affinements itératifs ont été apportés (réévaluation et d'optimisation progressive) au fur et à mesure, a conduit à un niveau de précision remarquable.

Annexes

1-Architecture du modèle



Références

[1] T. Cazenave, https://www.lamsade.dauphine.fr/~cazenave/index.php.

[2] T. Cazenave, "Improving Model and Search for Computer Go". IEEE Conference on Games 2021. ImprovingModelAndSearchForComputerGo.pdf