Université de Montpellier

Rapport du projet
Classification d'images

El Houiti Chakib
Kezzoul Massili

9 décembre 2021

1 Préparation des données

Cette partie concerne le prétraitement des données avant de les passer à un réseau de neurones. En effet, avant de rentrer dans le vif du sujet, de petites préparations s'impose. D'abords, les images contiennent des pixels qui ont comme valeurs un entier entre 0 et 255. Afin d'accelerer la convergence du réseau, il est primordiale de normaliser les données afin de les ranger dans une échelle de 0 à 1. Cela réduit grandement la complexité et le temps de calcul. Ensuite, nous transformant les étiquettes du jeu de données d'une chaine de caractère à du *one hot enconding*.

2 Construction des modèles

2.1 Première version

Pour commencer, nous avons défini un premier modèle simple. Celui-ci est composé de trois couches convolutionnels dont chaqu'une est suivie d'une couche de *MaxPooling*. Ces trois couches, ont respectivement 32, 64 et 128 filtres de sotie. Cette première partie constitue l'étape de *feature extraction*. Ensuite, viens la deuxième partie du réseau qui est constituée d'une couche *Flatten* qui applatit le vecteur de sortie de la convolution. Au final, une couche *dense* de 256 neurones viens apprendre de ces *features* pour enfin donner un résultat en sortie.

Ensuite, nous avons défini les hyperparamètres du modèle, notamment :

— l'optimizer utilisé, ici *Adam*
— le learning rate utilisé, ici *0.005*
— la fonction de perte utilisée, ici *mean_squared_error*
— le nombre d'epochs, ici de *5*
— le batch size, ici de *128*

L'entrainement de ce modèle avec ces paramètres là, nous donne ce résultat :

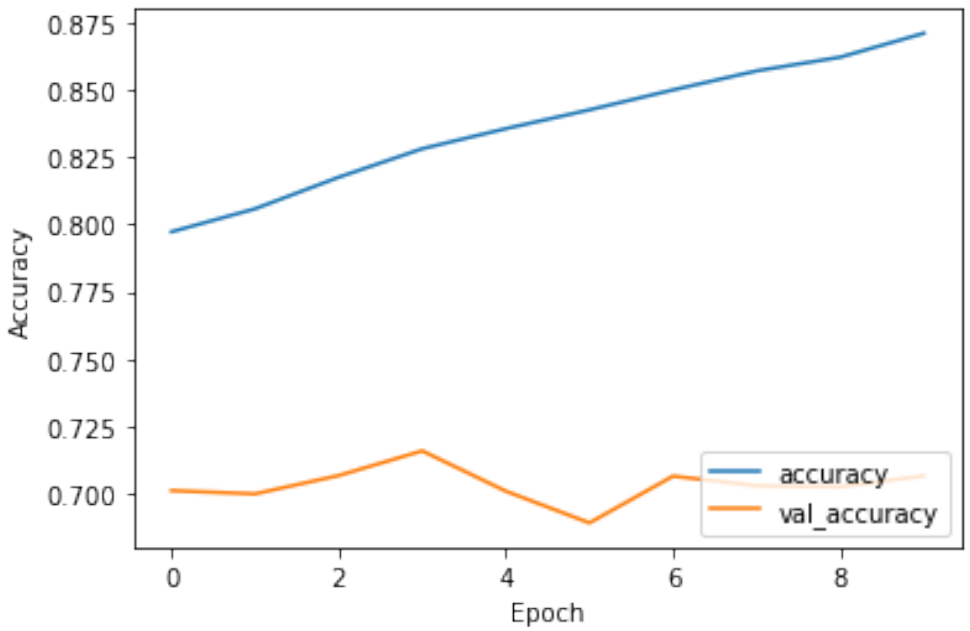

FIGURE 1 – Premier résultat

On voit ici que l'accuracy augmente progressivement jusqu'à atteindre environ 87% tandis que l'accuracy sur les données de validation stagne au alentours de 70%. On remarque donc du surapprentissage

par notre modèle. Nous avons donc ajouté une couche de *Dropout* afin de perturber le réseau lors de son entrainement. On voit sur le graphe ci-dessous que l'accuracy de validation se rapproche beaucoup plus de celle de l'apprentissage. Néaumoins, ces deux valeurs ne dépassent pas les 70% déjà atteinte par l'accuracy de validation lors du premier cas.

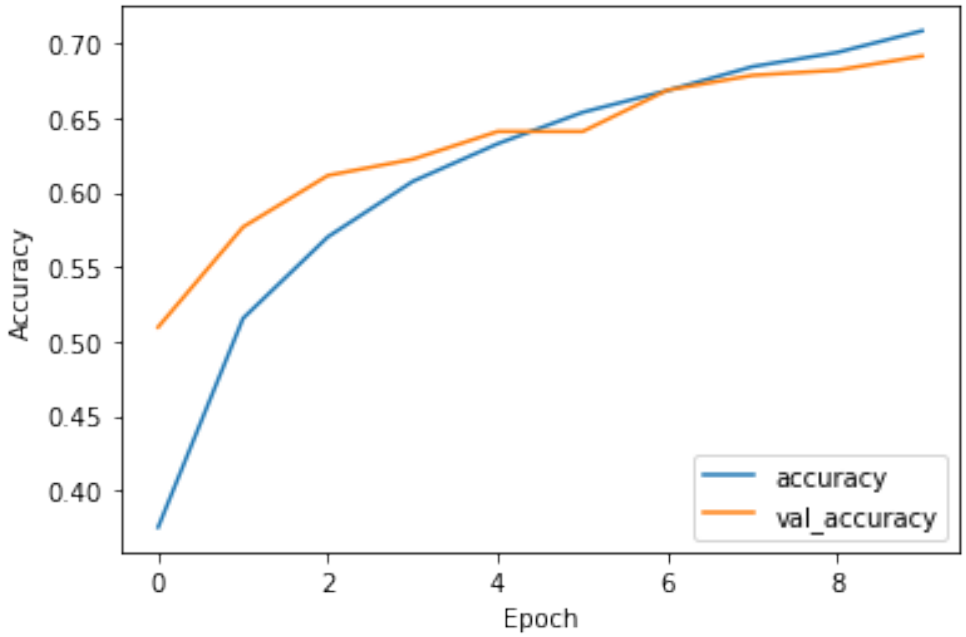

FIGURE 2 – Ajout d'une couche de *Dropout*

2.2 Améliorations de la structure

Nous avons commencer par ajouter une couche de *BashNormalization* après chaque couche de convolution ainsi que les couches *denses*. Ceci afin de garder les valeurs de sorties de ces couches avec une moyenne égale à 0 et un écart type de 1. Ceci rends le modèle beaucoup moins sensibles au *outliers*. En plus de ça nous avons doublé le nombre de couche de convolution afin d'avoir un plus large modèle avec plus de paramètres. Nous avons notamment, ajouté dans la deuxième partie du modèle une couche *dense* de 512 neurones.

Pour l'entrainement de ce modèle, nous avons gardé les même hyperparamètres sauf pour le nombre d'epochs que nous avons mis à 100. Ceci car il faut plus de temps au modèle pour converger vers une valeur. Nous obtenons le graphe ci-dessous :

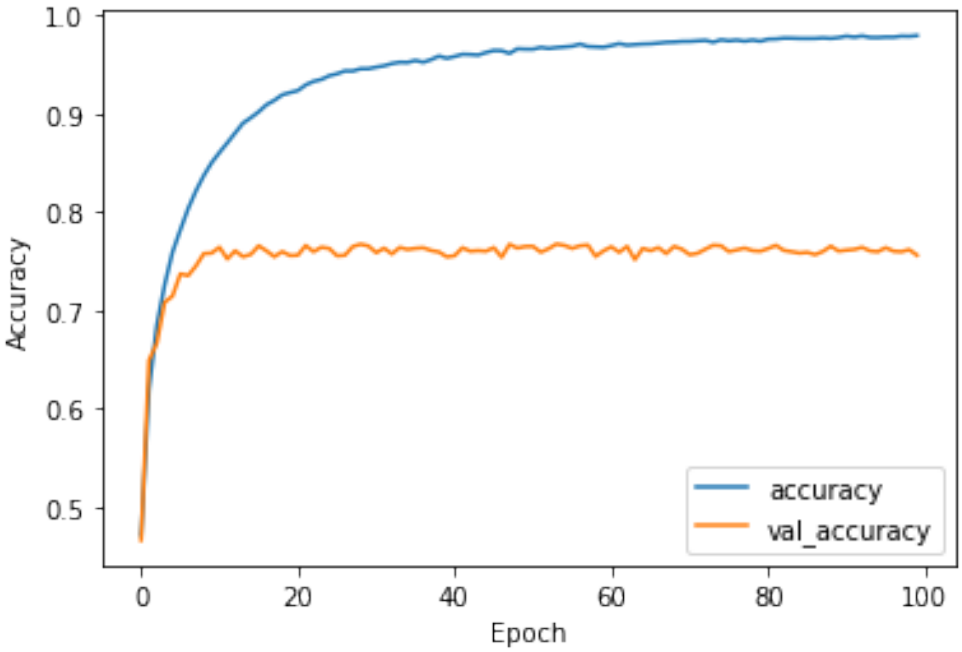

FIGURE 3 – Réseau de plus grande taille

Nous observons ici qu'avec un plus grand nombre d'epochs, on voit bien que ce n'est qu'a partir de la 20ème que l'accuracy commmence à se stabiliser. Mais, on remarque aussi qu'on retombe dans un cas de surapprentissage. En effet, la validation atteint difficilement les 76% tandis que l'accuracy de l'apprentissage va jusqu'a 98%. Il s'agit donc maintenant d'essayer d'optimiser notre modèle.

3 Optimisation des résultats

3.1 Augmentation des données

La première technique d'optimisation que nous avons utilisé est l'augmentation des données. Cette dernière consiste à appliquer des modifications sur les images d'origines. Cela permet, dans un premier temps, d'avoir plus de données à disposition et dans un second temps, apporter de la diversité à nos données. C'est donc une façon d'éviter le surapprentissage.

Pour réaliser cette tâche, plusieurs options s'offrent à nous :

— Une rotation des images ;
— Retourner l'image verticalement ou/et horizontalement ;
— Décaler l'image de quelques pixels de droite à gauche ou/et de haut en bas.

Pour cela, nous avons réalisé plusieurs tests en variant ces paramètres, pour obtenir au final la meilleure combinaison possible de ces options pour notre modèle.

3.2 Variation des hyperparamètres

3.3 Transfert learning

4 Analyses des résultats

5 Conclusion et perspectives