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'imposent. D'abord, les images contiennent des pixels qui ont comme valeurs un entier entre 0 et 255. Afin d'accélérer la convergence du réseau, il est primordial 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 chaîne 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 convolutionnelles 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, vient la deuxième partie du réseau qui est constituée d'une couche *Flatten* qui aplatit 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'entraînement de ce modèle avec ces paramètres-là, nous donne ce résultat :

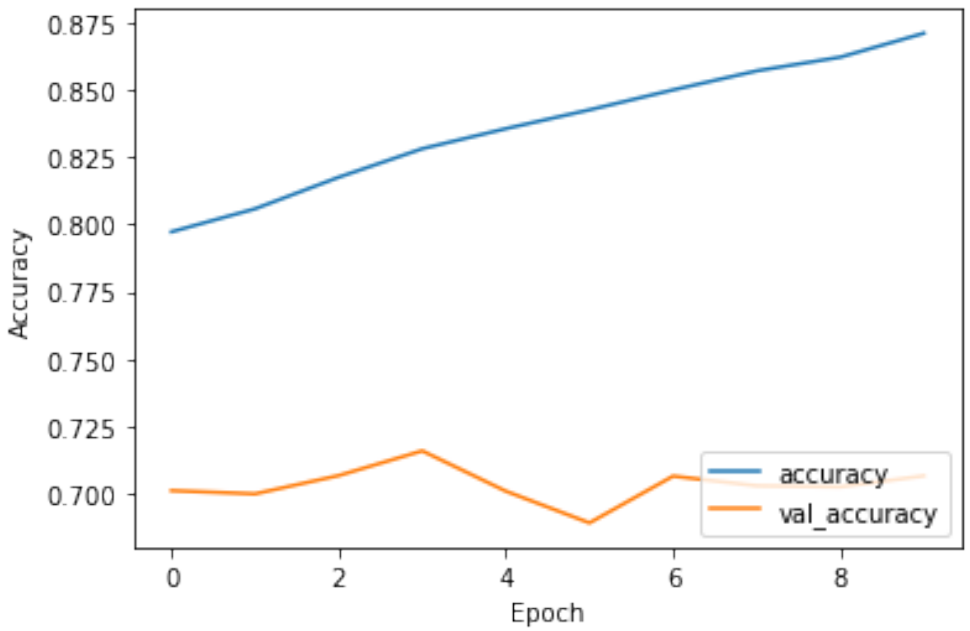


FIGURE 1 – Premier résultat

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

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

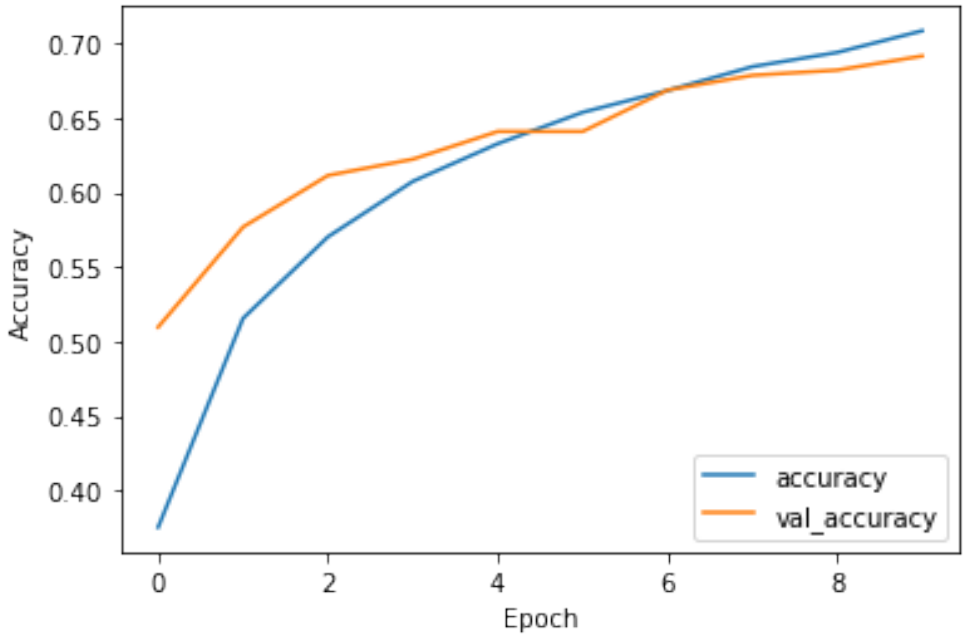


FIGURE 2 – Ajout d'une couche de *Dropout*

2.2 Améliorations de la structure

Nous avons commencé 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 rend le modèle beaucoup moins sensible au *outliers*. En plus de ça, nous avons doublé le nombre de couches de convolution afin d'avoir un plus large modèle avec plus de paramètres. Nous avons notamment, ajoutés dans la deuxième partie du modèle une couche *dense* de 512 neurones.

Pour l'entraînement 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 :

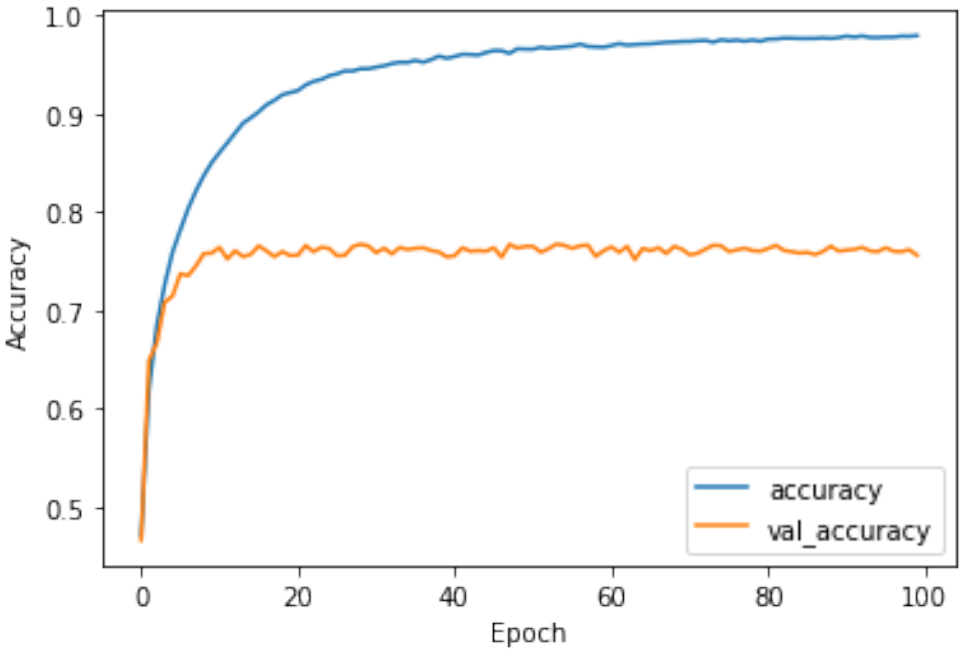


FIGURE 3 – Réseau de plus grande taille

Nous observons ici qu'avec un plus grand nombre d'epochs, on voit bien que ce n'est qu'à partir de la 20ème que l'accuracy commence à se stabiliser. Mais on remarque aussi qu'on retombe dans un cas de sur apprentissage. En effet, la validation atteint difficilement les 76% tandis que l'accuracy de l'apprentissage va jusqu'à 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 utilisée 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 sur apprentissage.

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. Nous avons obtenu les résultats suivants :

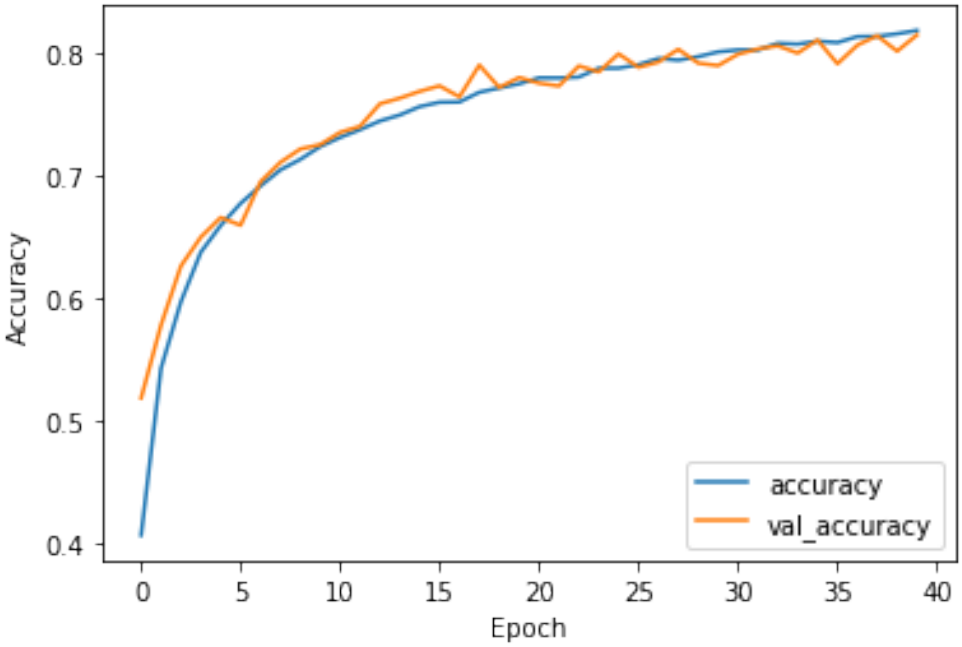


FIGURE 4 – Data augmentation

Comme on peut le remarquer, il n'y a plus de sur apprentissage avec une accuracy de validation très proche de celle de l'apprentissage qui valent environ 82%. Ce qui est déjà une très nette amélioration.

3.2 Variation des hyperparamètres

La deuxième approche consiste à trouver les meilleurs hyperparamètres pour notre modèle. Nous avons donc défini pour chaqu'un une liste de valeurs. Puis, on a testé toutes les combinaisons possibles entre ces hyperparamètres. Nous avons obtenu le tableau de la Figure 5.

On déduit de ce tableau que les meilleurs paramètres sont ceux de la ligne coloré en bleu.

En appliquant, ces hyperparamètres et les meilleures options de la data augmentation, on obtient, le graphe de la Figure 6 :

On remarque clairement l'amélioration de notre de modèle en terme de valeur d'accuracy de la validation qui est proche de 90%. Ce résultat est obtenu en appliquant les deux méthodes d'optimisations.

4 Transfert learning

Dans cette partie du projet, on a fait du *Transfert learning* qui consiste à utiliser des modèles pré-entraînés. Il existe deux techniques de *Transfert learning* :

Feature extraction Une technique, se basant sur le fait de gelé toutes les couches convolutionnels et de créé un small classifieur qui récupère les Features extraites de la dernière couche.

Fine tunnig Cette technique, est faite pour affiner les résultats, elle consiste à gelé un nombre précis de couches, ensuite à dégelé la dernière partie des couches, pour pouvoir mettre à jour les poids des sorties de couches lors de l'entraînement.

Pour réaliser cela, on a choisi trois modèles :
— MobileNetV2
— DenseNet121
— VGG16

LOSS	LR	DROPOUT	OPT	POOLING	ACC	VAL_ACC	LOSS_VALUE	VAL_LOSS
categorical_crossentropy	0.0050	0.3	adam	max	0.982699990272522	0.8134999871253967	0.05094645917415619	0.8739220499992371
mean_squared_error	0.0050	0.3	adam	max	0.9755399823188782	0.8162999749183655	0.0039016399532556534	0.028774920850992203
categorical_crossentropy	0.0010	0.3	adam	max	0.9873999953269958	0.8122000098228455	0.036502156406641006	0.9084367156028748
mean_squared_error	0.0010	0.3	adam	max	0.9668800234794617	0.8054999709129333	0.005121569149196148	0.030964171513915062
categorical_crossentropy	0.0001	0.3	adam	max	0.9850999712944031	0.6791999936103821	0.05323021486401558	1.4379479885101318
mean_squared_error	0.0001	0.3	adam	max	0.9637600183486938	0.6966000199317932	0.006020405329763889	0.0457441508769989
categorical_crossentropy	0.0050	0.5	adam	max	0.9847599864006042	0.8166999816894531	0.04526571184396744	0.840005099773407
mean_squared_error	0.0050	0.5	adam	max	0.9549199938774109	0.8058000206947327	0.006954308599233627	0.030467113479971886
categorical_crossentropy	0.0010	0.5	adam	max	0.9773200154304504	0.8014000058174133	0.06537435203790665	0.9227089881896973
mean_squared_error	0.0010	0.5	adam	max	0.9673799872398376	0.8029000163078308	0.00509100966155529	0.03191027790307999
categorical_crossentropy	0.0001	0.5	adam	max	0.9894400238990784	0.7031000256538391	0.0506577305495739	1.1684964895248413
mean_squared_error	0.0001	0.5	adam	max	0.9255399703979492	0.7019000053405762	0.011594465002417564	0.04377572610974312
categorical_crossentropy	0.0050	0.3	adam	avg	0.9678000211715698	0.809499979019165	0.09721206873655319	0.9040825366973877
mean_squared_error	0.0050	0.3	adam	avg	0.9700599908828735	0.8172000050544739	0.00472243782132864	0.029285984113812447
categorical_crossentropy	0.0010	0.3	adam	avg	0.9852799773216248	0.8251000046730042	0.04228955879807472	0.8813745975494385
mean_squared_error	0.0010	0.3	adam	avg	0.9795600175857544	0.8328999876976013	0.00328009482473135	0.026815585792064667
categorical_crossentropy	0.0001	0.3	adam	avg	0.9963799715042114	0.734499990940094	0.025248408317565918	1.082409143447876
mean_squared_error	0.0001	0.3	adam	avg	0.9676200151443481	0.7422999739646912	0.005355002824217081	0.038527511060237885
categorical_crossentropy	0.0050	0.5	adam	avg	0.9815800189971924	0.823199987411499	0.053745608776807785	0.8203244209289551
mean_squared_error	0.0050	0.5	adam	avg	0.939740002155304	0.7971000075340271	0.009217387065291405	0.032532718032598495
categorical_crossentropy	0.0010	0.5	adam	avg	0.983020007610321	0.823199987411499	0.04948342591524124	0.8062456250190735
mean_squared_error	0.0010	0.5	adam	avg	0.9724400043487549	0.8241000175476074	0.004304856061935425	0.027826843783259392
categorical_crossentropy	0.0001	0.5	adam	avg	0.9838200211524963	0.7491000294685364	0.07062611728906631	0.9248133897781372
mean_squared_error	0.0001	0.5	adam	avg	0.9383400082588196	0.7493000030517578	0.00979839637875557	0.03647753223776817

FIGURE 5 – Table des hyperparamètres

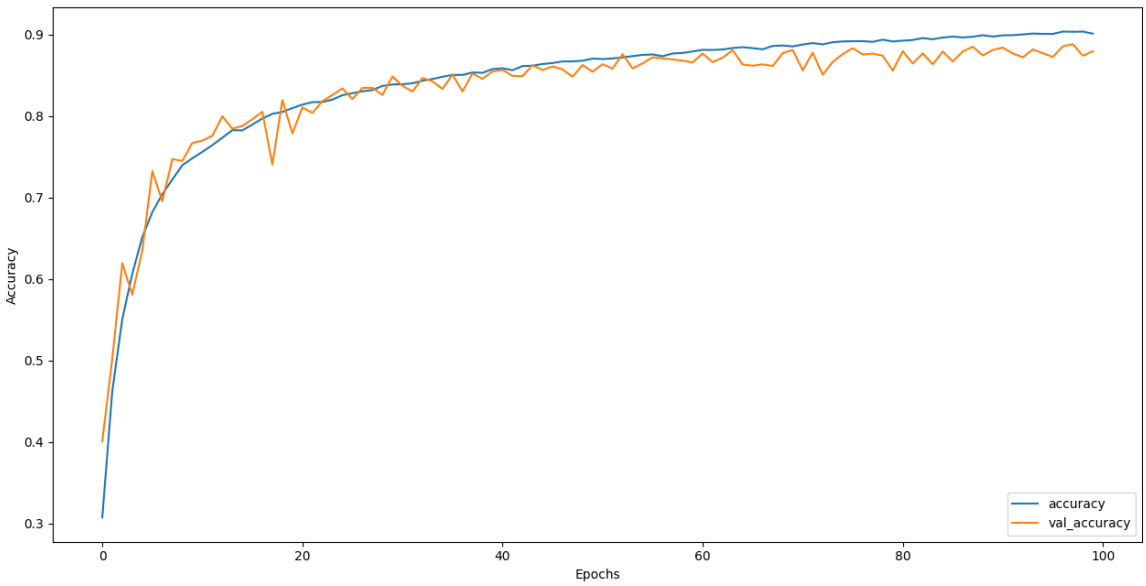


FIGURE 6 – Final modèle

On a obtenu le résultat de la Figure 7 en comparant ces derniers, sur le nombre de paramètres qui peuvent prendre et le score atteint par ces modèles sur nos données.

Les deux techniques peuvent être utilisées séparément ou aussi en les combinant. Donc on a choisi de prendre le meilleur modèle entre les trois listés au-dessus pour les comparer dans leur deux cas d'utilisation. Une figure 8 montrant une comparaison entre les deux techniques.

On remarque que les deux techniques sont presque similaires, vues qu'on a dégelé que 28 sur 428

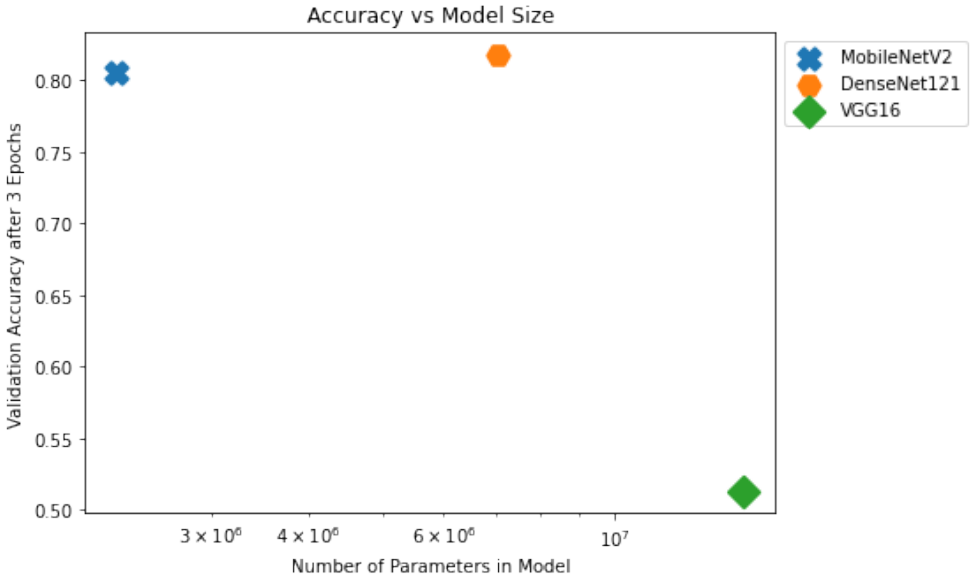
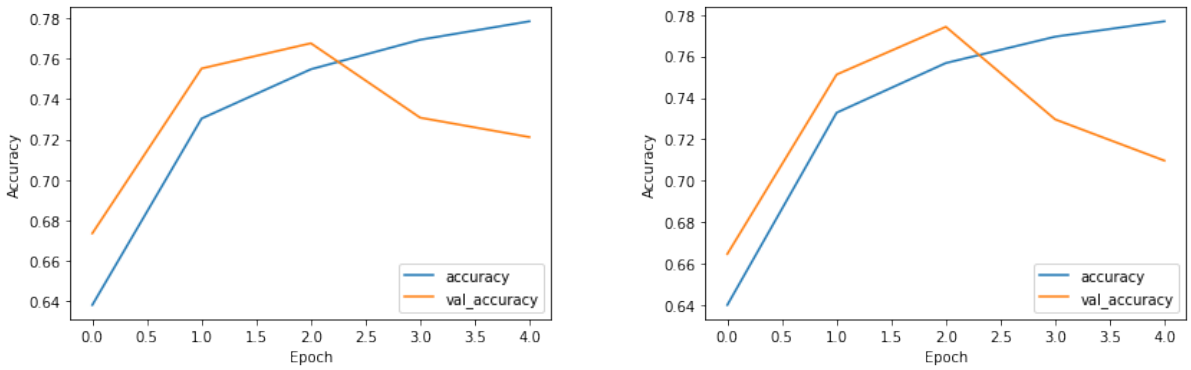


FIGURE 7 – Comparaison des modèles pré-entraînés



(a) Feature extraction

(b) Fine Tunning

FIGURE 8 – Comparaison entre les deux techniques

couches lors du *Fine tunning*, ce qui n'est pas assez signifiant.

La combinaison des deux techniques, consiste à commencer par du *Feature extraction*, ensuite ré entraîner le modèle avec du *Fine tunning* comme le montre la figure 9

On ne remarque pas de très bons résultats, car on a pas fait de l'augmentation de données avec le *Transfert learning*.

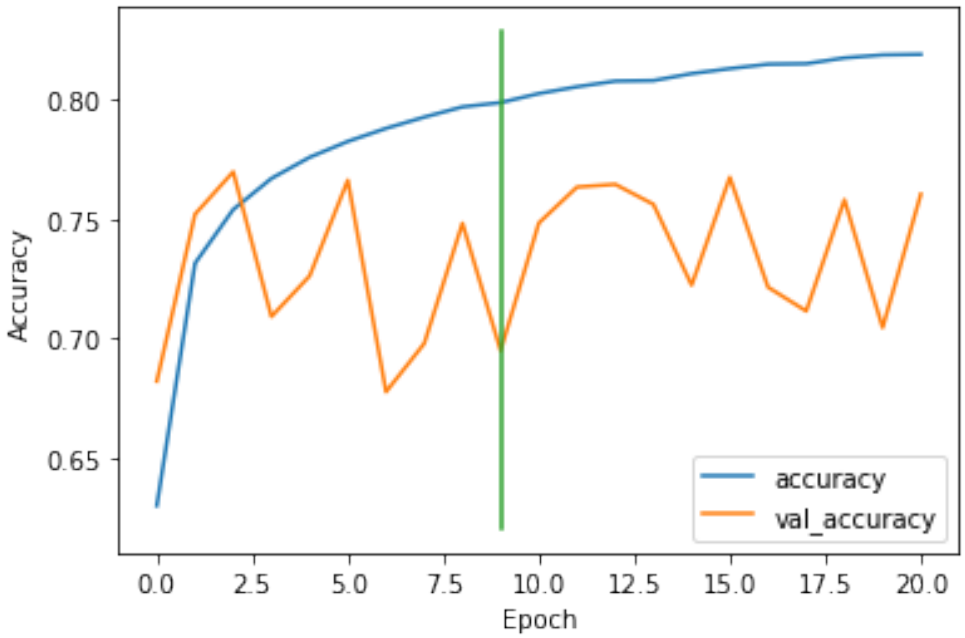


FIGURE 9 – Combinaison des deux techniques

5 Conclusion et perspectives

Ce Projet nous a permis de comprendre au mieux le fonctionnement des réseaux de neurones, plus précisément les modèles convolutionnels et les différentes techniques pour améliorer ces modèles tel que :

- Data augmentation.
- Variation des hyperparamètres pour obtenir la meilleure combinaison de ces derniers.
- Transfert learning.

5.1 Perspectives

Les CNNs demandent des ressources de calcul qui nous ont freinés lors de notre expérimentation avec ces modèles. On aurait voulu pouvoir par exemple :

- Varier plus d'hyperparamètres et d'options de data augmentation et pouvoir aussi les varier au même temps.
- Exploiter le transfert learning avec de la data augmentation.