**8INF804 - Vision artificielle et traitement des images - Hiver 2023**

Professeur Julien Maitre, Ph.D.

**UNIVERSITÉ DU QUÉBEC À CHICOUTIMI**

Département d’Informatique et de Mathématique

**Comparaison des réseaux de neuronnes convolutifs**

**BALB10020109 - Becaye Baldé**

Comparaison des réseaux de neuronnes convolutifs

[**Comparaison des réseaux de neuronnes convolutifs 2**](https://docs.google.com/document/d/1NLd-n_o-Kr1aGvikscBJeYIIl6chqd1Sd8WecWQZUIk/edit#heading=h.h3cfp3d9f1jp)

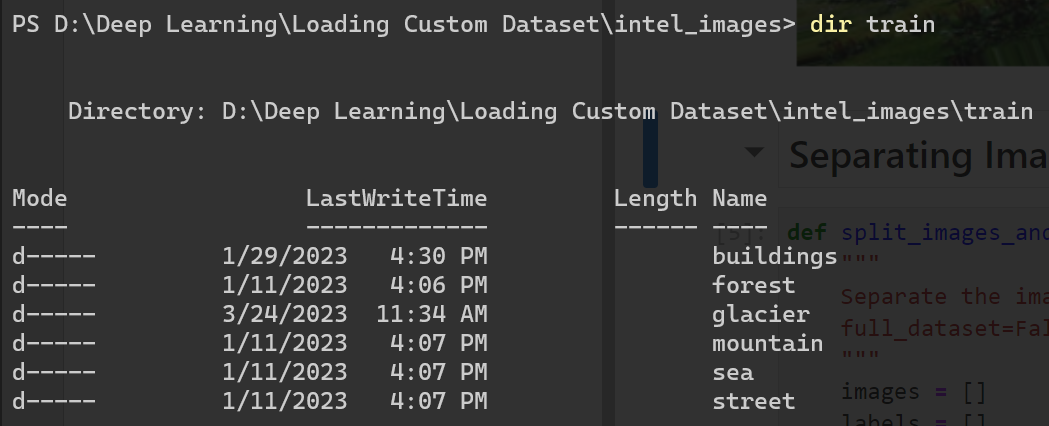
Dans ce TP, j’au testé des réseaux de convlutifs: Un CNN personnel, et deux  models  state-of-the-art (ResNet-50 et VGG16).

J’ai utilisé l’ensemble de données: [intel images](https://www.kaggle.com/datasets/puneet6060/intel-image-classification) de Kaggle. Ce dataset contient 5 categories d’images: batîments, forêts, mers, glaciers et rue. Ces images sont reparties en deux ensembles : 14034 images pour l’entraînement et 3000 pour la validation.

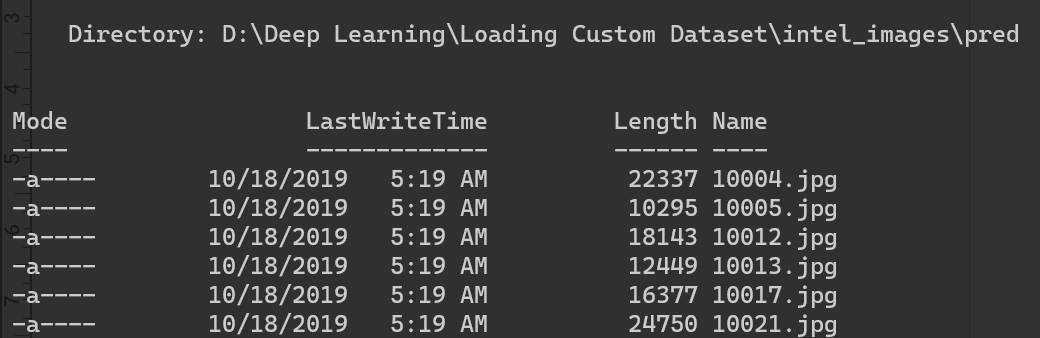
Ces images sont reparties en 3 dossiers:

* Train: contient les images d’entraînement ;
* Test: contient les images de validation ;
* Pred: contient des images de test.

Les images dans les deux premiers dossiers sont regroupés selon leur classes:



Et pred contient des images non annontées :



Voici quelques images de l’ensemble de données:



On peut voir qu’elles sont très diversifées. Cependant le dataset est impure et contient des images mal annotées. Par example, la fleur suivante est annotée “glacier”:



Passons donc à l’entrainement du réseau de neuronnes.

1. CNN Personalisé

1.1. Modèle initial

J’ai d’abord commencé avec une architecture très simple:

* 2 couches de convolution
* 2 couches de max pooling

Cependant, le model était en surapprentissage.

TODO - CNN

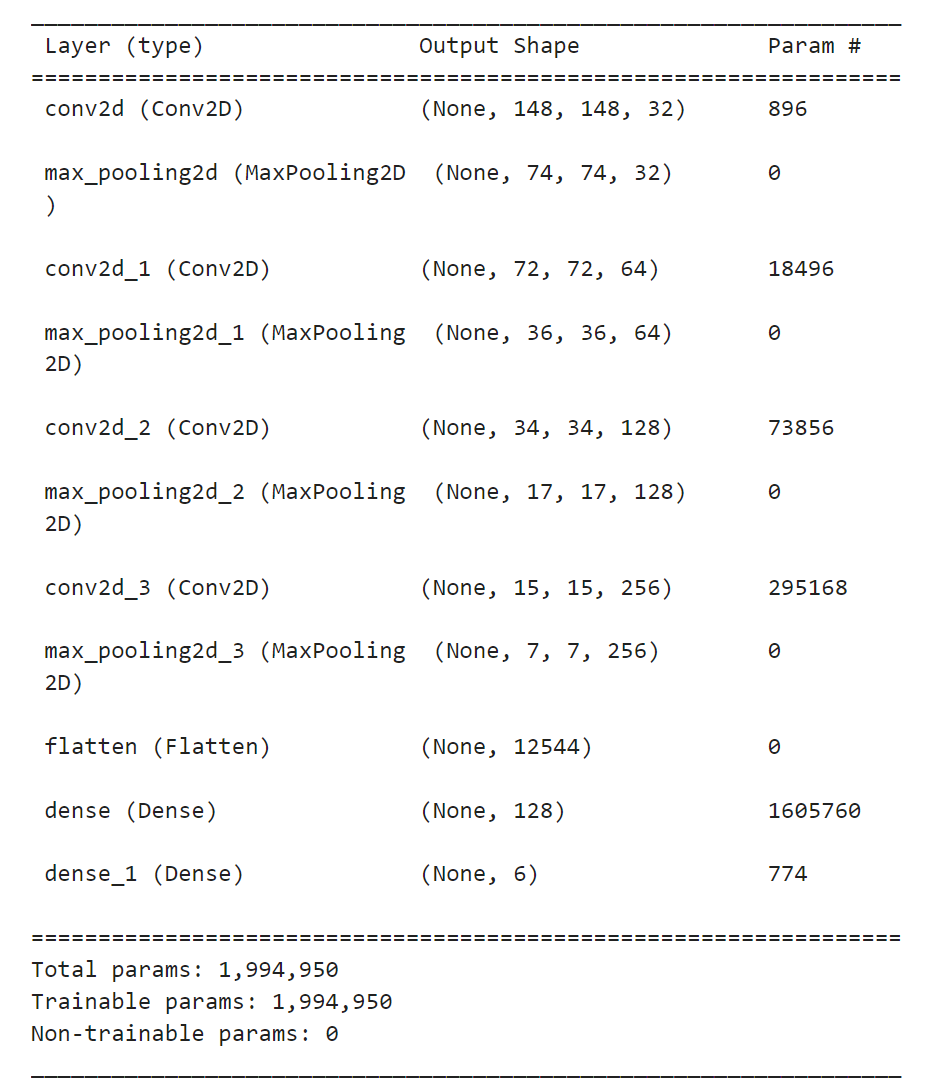
J’ai donc décidé de le rendre plus profond:

1.2. Modèle final

1.2.1. Architecture du modèle

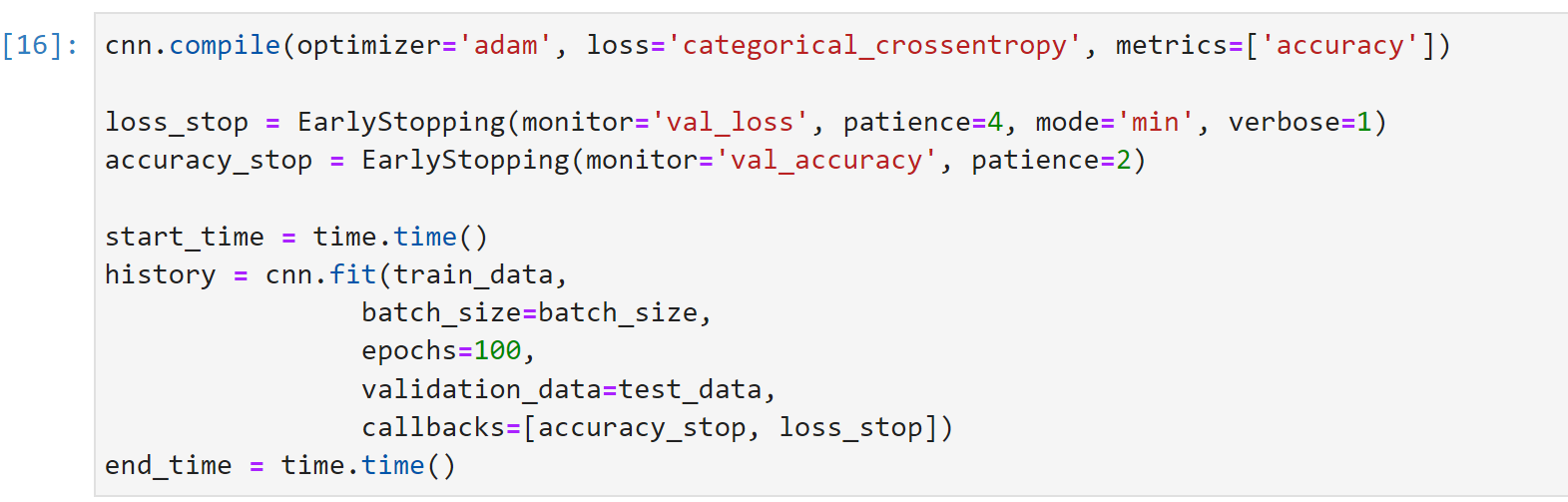
Le CNN final contient :

* 4 couches de convolution ;
* 4 couches de max pooling ;

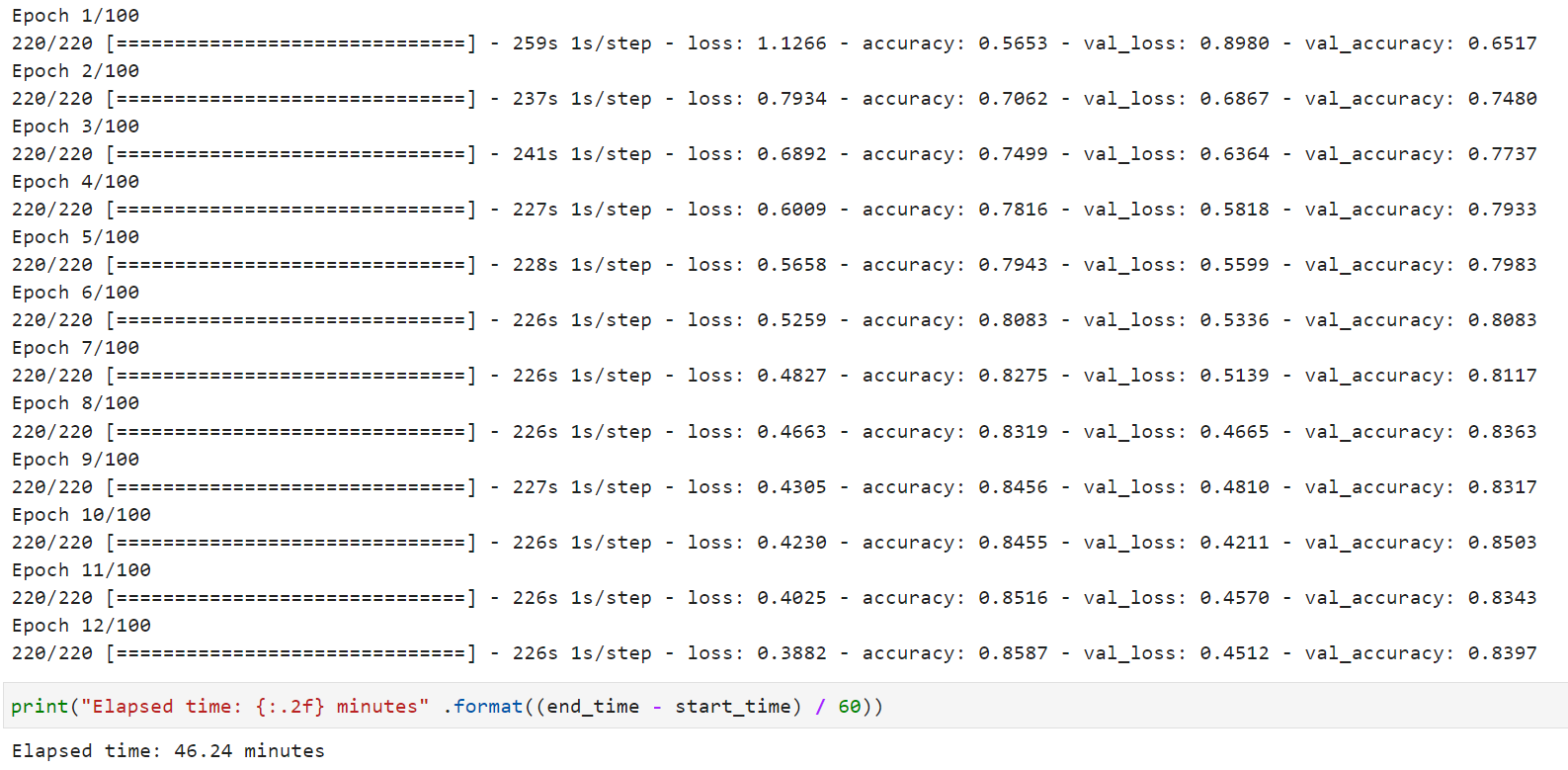


1.2.2. Entraînement du modèle

Puis, j’ai entraîné le modèle pour un nombre maximal de 100 epochs, avec des conditions d’arrêt au cas où l’accuracy ou la perte ne s’améliorerait pas.

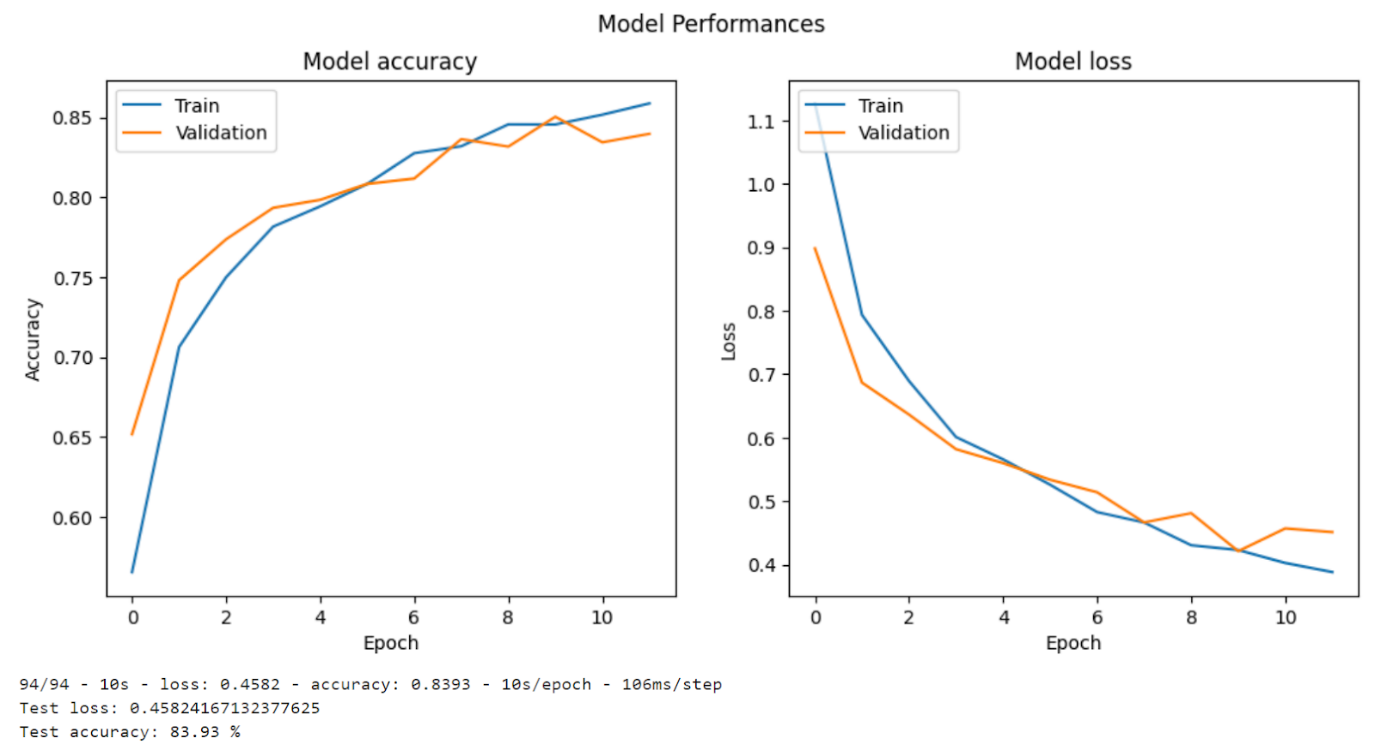


Après seulement 12 epochs, le modèle a commencé s’est stabilisé:



1.2.3. Evaluation des performances

Finalement, on obtient des performances similaires sur les données d’entraînement et de validation (83% à peu près):



1.2.4. Prediction d’une nouvelle image

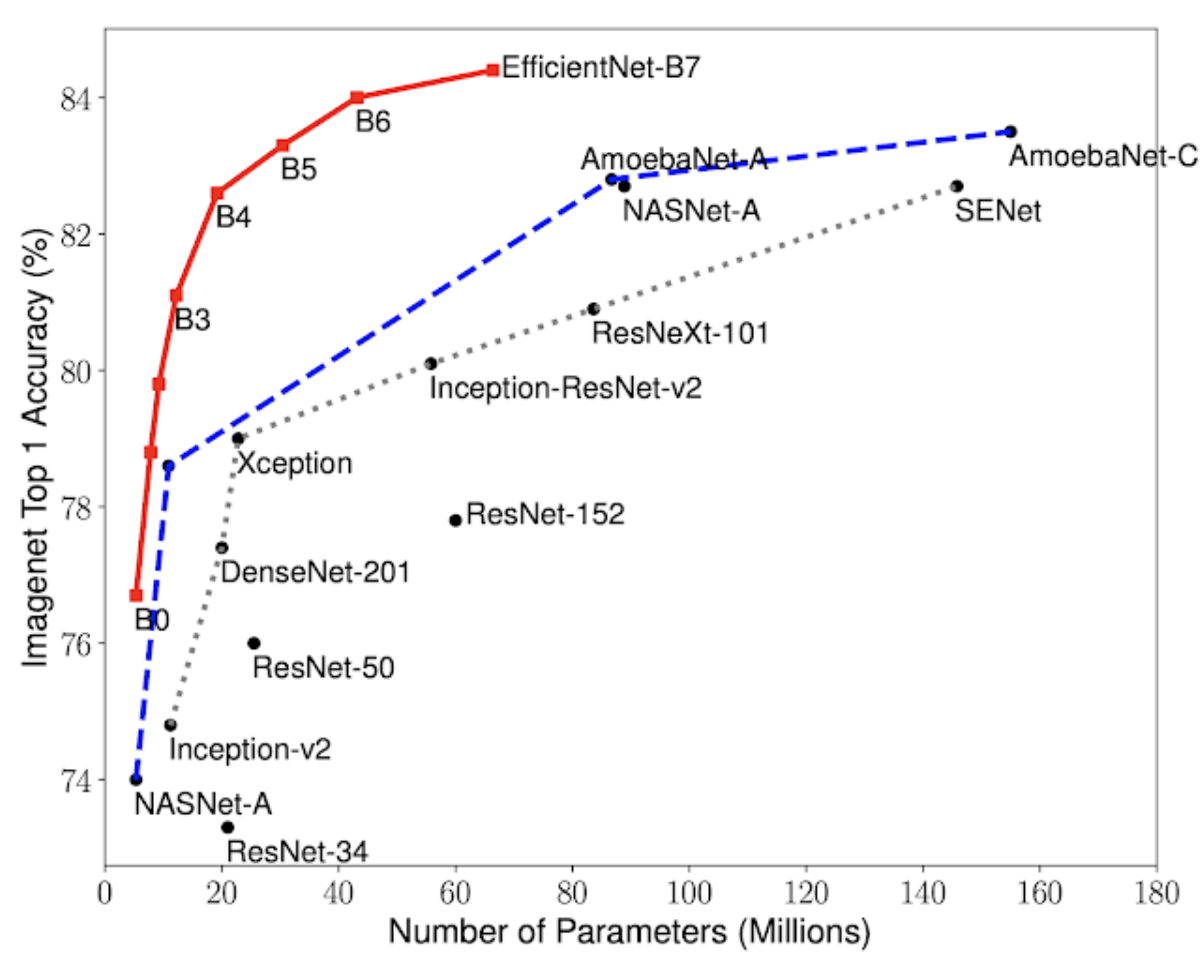
Lorsqu’on lui envoit cette image d’une forêt, le modèle est à 99,93% sûr que c’est une forêt, plutôt pas mal !



Maintenant comparons les performances de notre CNN avec ceux d’un modèle state-of-the-art.

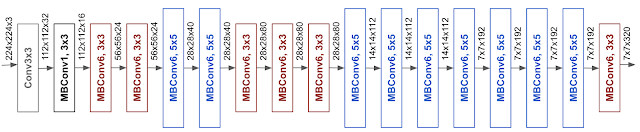
1. EfficientNet

J’ai choisi le modèle pré-entrainé EfficientNet. [EfficientNetV2L](https://keras.io/api/applications/efficientnet_v2/#efficientnetv2l-function) est le modèle le plus performant une accuracy (justesse) de 97.5% (source: [keras.io](https://keras.io/api/applications/efficientnet_v2/#efficientnetv2l-function)).

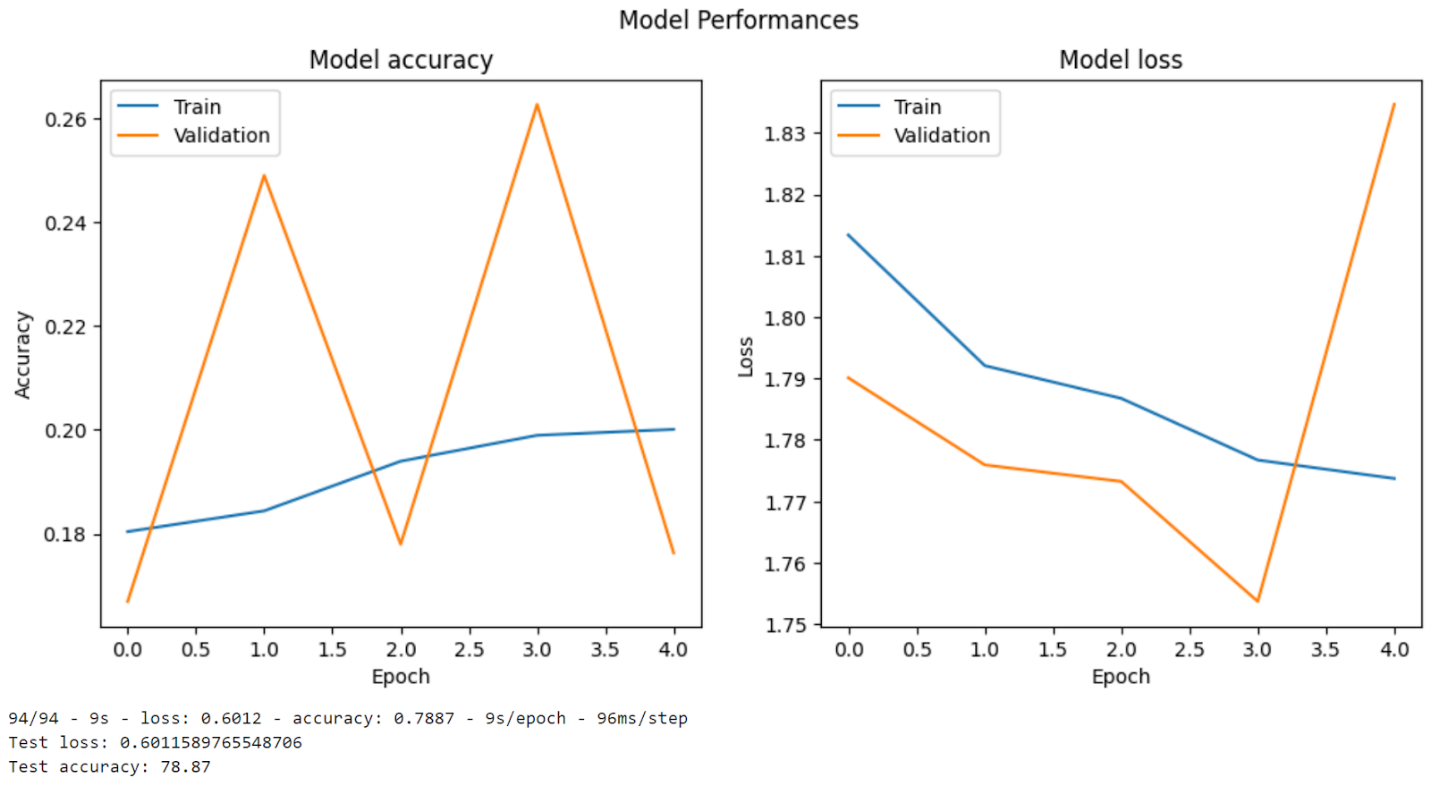


Source: ai.googleblog.com

Il introduit le “compound scaling”, ce qui améliore l’accuracy.



Source: ai.googleblog.com

Pour revenir au TP, EfficientNet a produit des résultats avec une accuracy très basses: moins de 20%. Le modèle est très instable et la fonction perte augmente avec le nombre d’epochs.

Conclusion

Ce TP m’a permis d’avoir une expérience pratique sur les réseaux de neuronnes convolutionnels. Un des points les plus importants que j’ai retenu est l’importance des données.

En expérimentant, évidement, plus les données d’entraînement était nombreuses, plus le modèle était précis. Mais la qualité des données est au moins aussi importante. Je pense que le fait d’avoir un dataset pas très limpide a affecté les performances de mon modèle. Hélàs, cela m’aurait pris trop de temps de labeliser ces 17000 images manuellement.