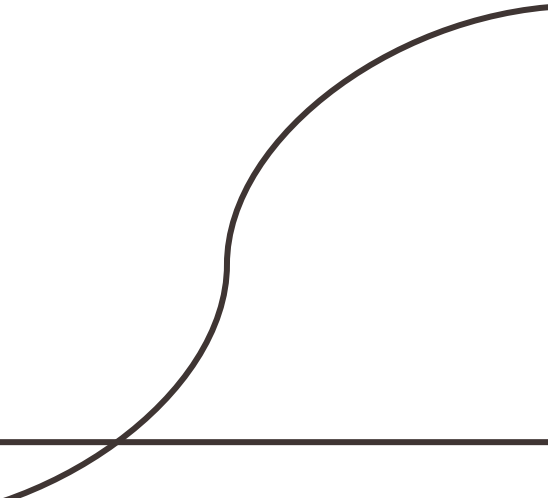

Classez des images à l'aide d'algorithmes de Deep Learning

Soutenance de projet

Nicolas FAUCONNIER
Parcours Ingénieur ML
01/08/2022



Plan

1.

Problématique

Contexte, missions et
dataset

2.

Modèle “from scratch”

VGG16 avec initialisation aléatoire
des poids et améliorations

3.

Transfert Learning

Essais de différents modèles
pré-entraînés

4.

Modèle Final

Hyperparameters et Fine
Tuning



1.

Problématique

Contexte, missions et dataset

Contexte

Une association locale de protection des animaux n'a pas le temps ni les ressources nécessaires pour référencer les images des nombreux pensionnaires passés par le refuge.

Missions

- Entraîner un algorithme capable de classer les images en fonction de la race du chien présent sur l'image
 - Développer un script Python fournissant la race du chien prédite par le modèle à partir d'une photo
-

Dataset



Stanford Dogs Dataset:

- 120 races de chiens
- 20580 images issues d'ImageNet

Limitation à 10 classes et 1919 images, afin de pouvoir effectuer les nombreux trainings (différentes architectures & hyperparamètres) dans des temps raisonnables

Split train/validation à 80/20, soit 1540 et 379 images

Pré-processing: resize 224x224


Utilisation de Google Colab et de **GPU** (Tesla P100) pour raccourcir les temps d'entraînement



2.

Modèle “from scratch”

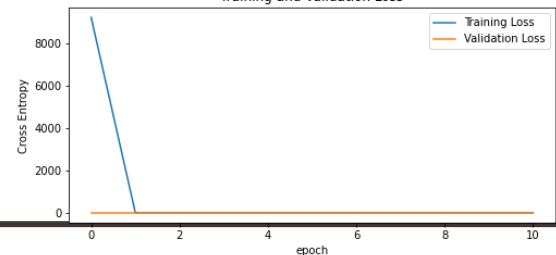
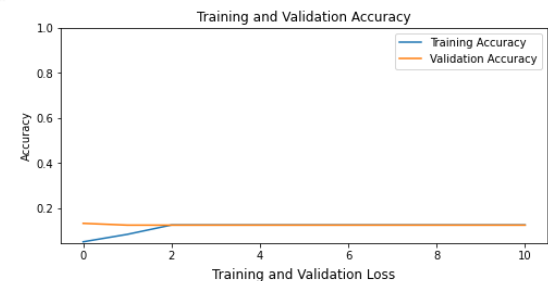
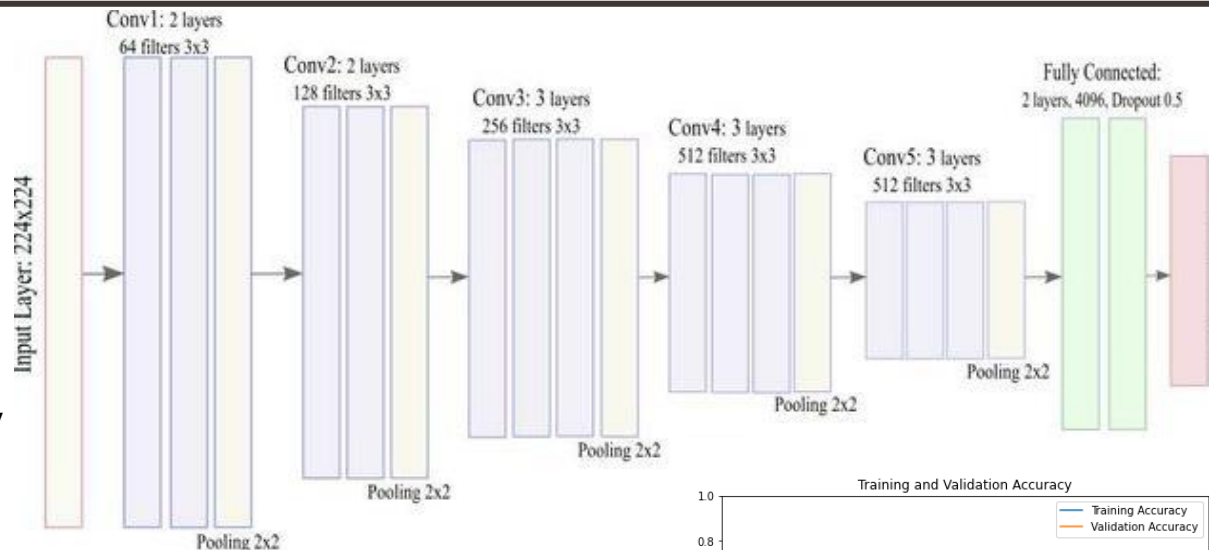
VGG16 avec initialisation aléatoire des poids et
améliorations



VGG16 Standard

Modèle de départ avec tentatives d'amélioration itératives

- Loss = categorical crossentropy
- Optimizer = Adam
- Learning rate = 0,001
- Activation= ReLu & Softmax
- Epochs = 50
- Utilisation d'Early Stopping (patience = 10)

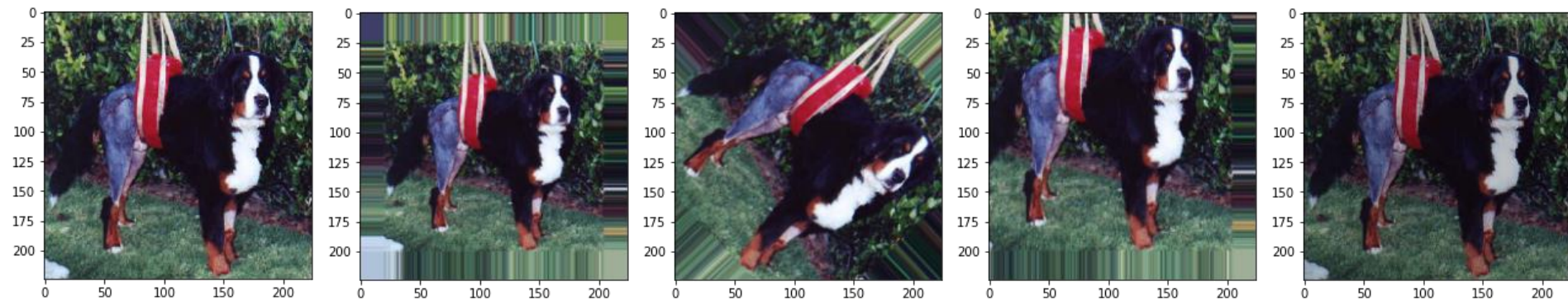


Accuracy Validation: 0,13

Nombre d'epochs: 11 (early stopping)

Temps d'entrainement: 3min 18s

Data Augmentation

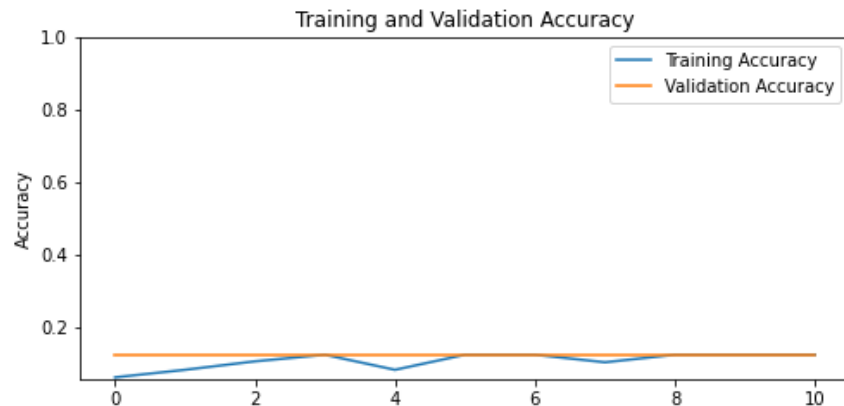


Effectué avec `keras.ImageDataGenerator`

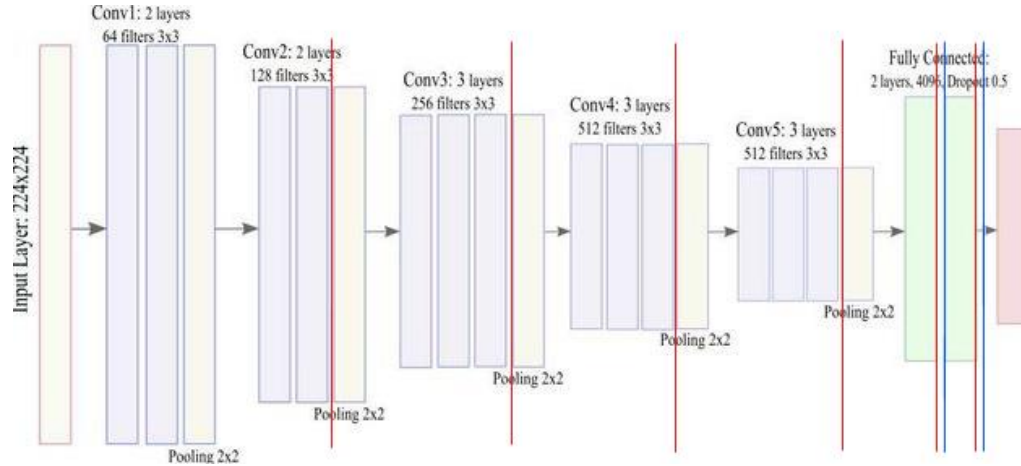
Accuracy Validation: 0,12

Nombre d'epochs: 11 (early stopping)

Temps d'entrainement: 6min 35s



Ajout de Batch Normalization & Dropout

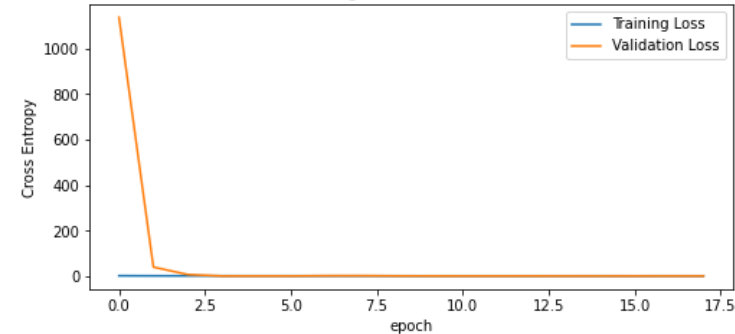
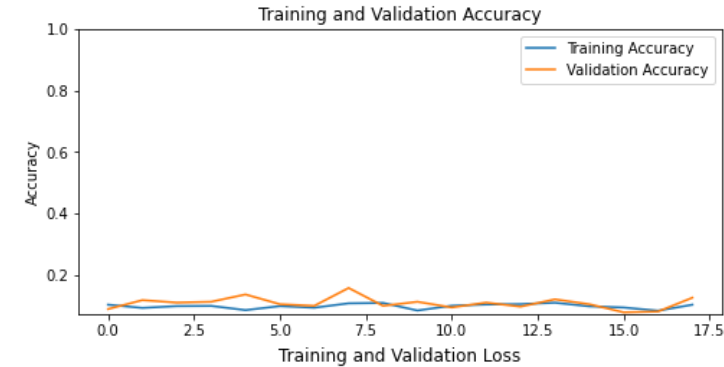


- couches de **Batch Normalization**
- couches de **Dropout** (rate = 0,5)

Accuracy Validation: 0,16

Nombre d'epochs: 18 (early stopping)

Temps d'entrainement: 11min 24s



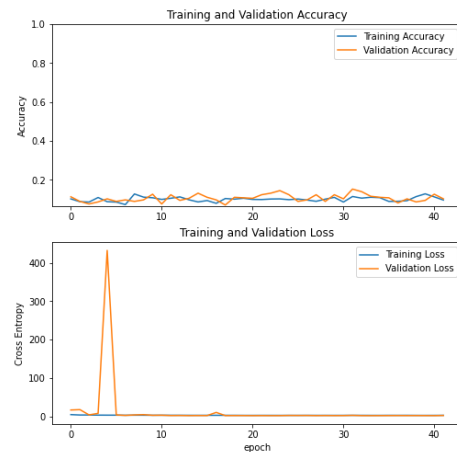
Optimizer et Kernel size

RMSprop

Accuracy Validation: 0,15

Nombre d'epochs: 42 (early stopping)

Temps d'entrainement: 26min 22s

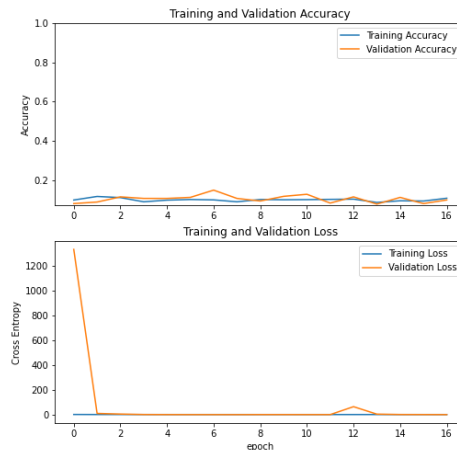


Kernel size = 5x5

Accuracy Validation: 0,15

Nombre d'epochs: 17 (early stopping)

Temps d'entrainement: 12min 7s



ResNet50

Architecture chargée depuis Keras

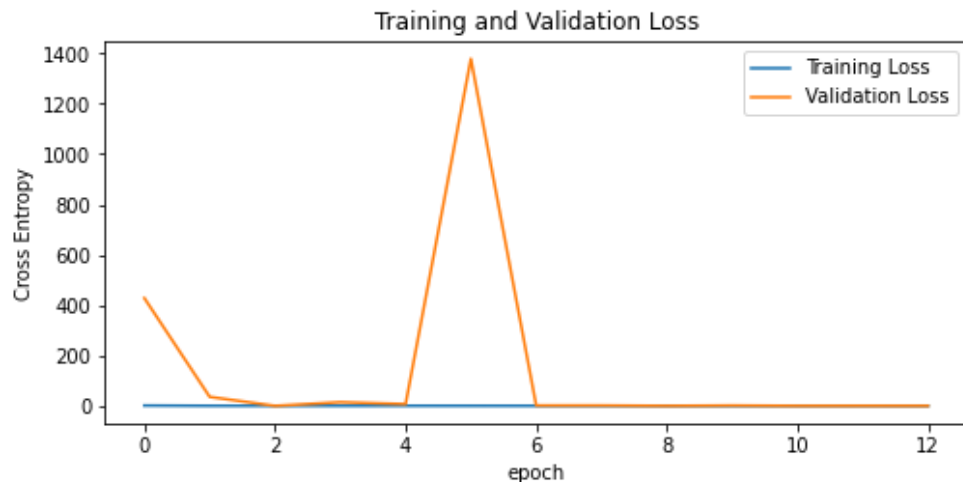
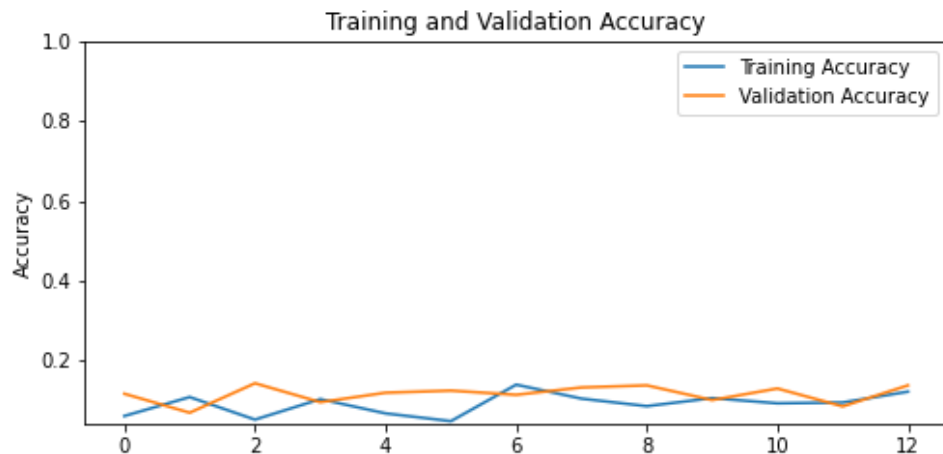
Initialisation aléatoire des poids

nb. pas de Transfert Learning

Accuracy Validation: 0,14

Nombre d'epochs: 13 (early stopping)

Temps d'entrainement: 8min 29s



3. Transfert Learning

Essais de différents modèles pré-entraînés

Couches supérieures communes

- Activation = ReLu & Softmax
- Dropout rate = 0,5
- Loss = categorical crossentropy
- Learning rate = 1e-4
- Optimizer = Adam

50 Epochs avec Early stopping

nb. Les couches inférieures sont figées

→ L'ajout de Dropout limite l'overfitting et améliore les performances (mais pas l'ajout de Batch Normalization)

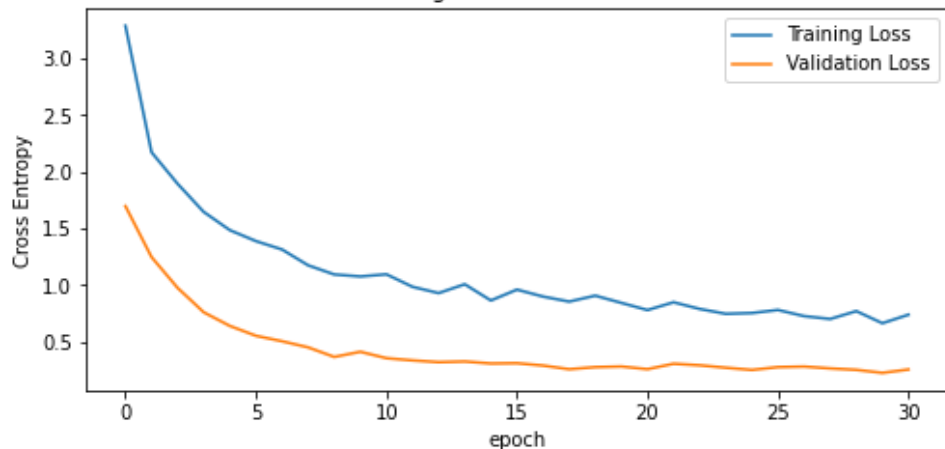
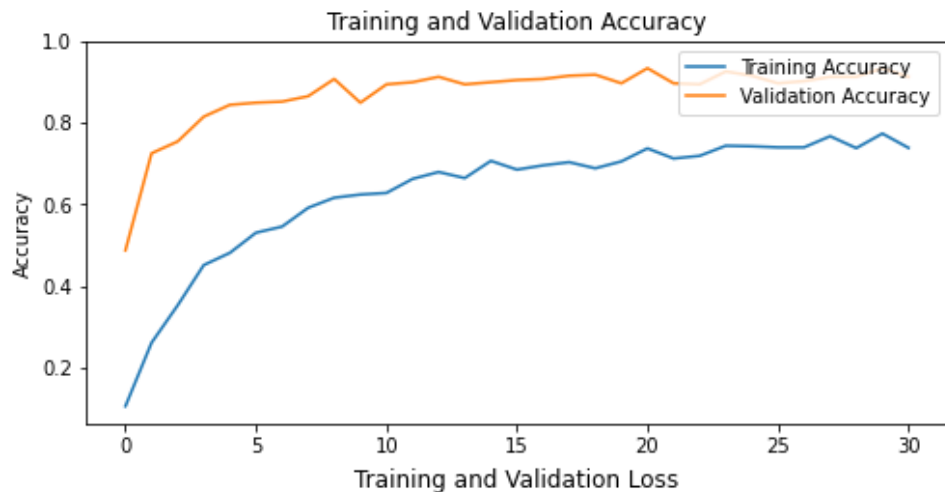
Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 2048)	23587712
flatten_7 (Flatten)	(None, 2048)	0
dense_19 (Dense)	(None, 1024)	2098176
dropout_6 (Dropout)	(None, 1024)	0
dense_20 (Dense)	(None, 512)	524800
dropout_7 (Dropout)	(None, 512)	0
dense_21 (Dense)	(None, 10)	5130

ResNet50 pré-entraîné

Accuracy Validation: 0,934

Nombre d'epochs: 31 (early stopping)

Temps d'entrainement: 19min 4s

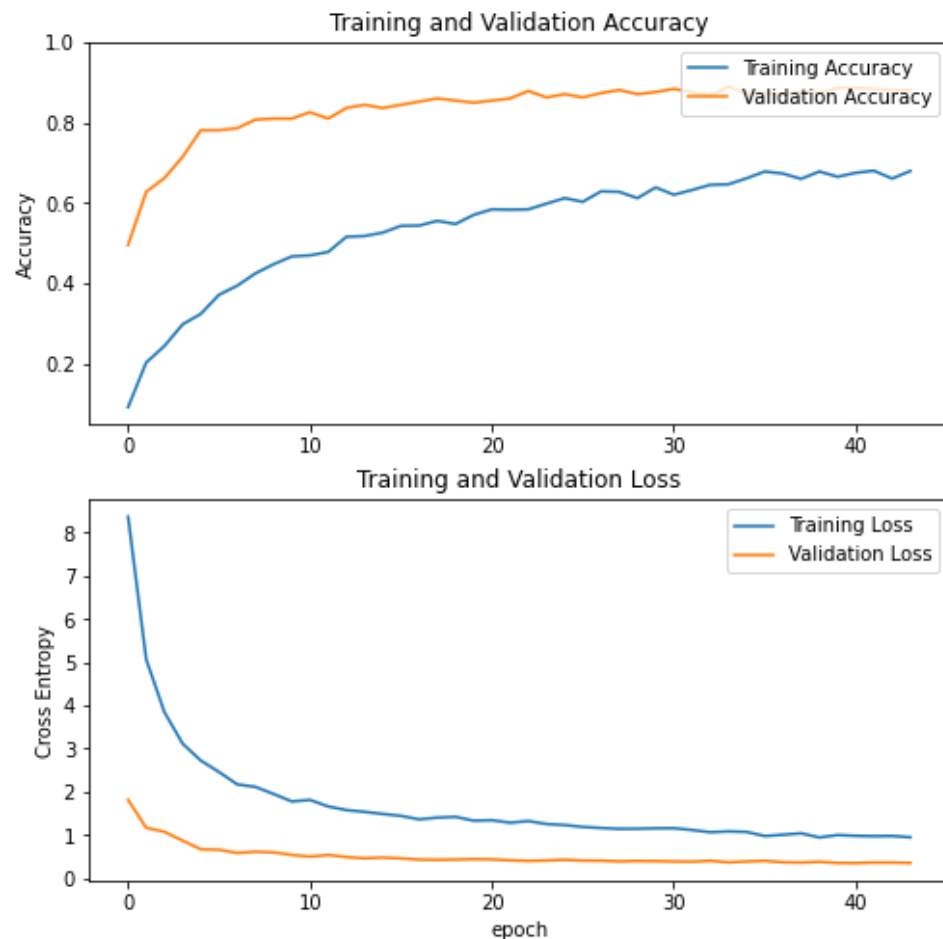


VGG16 pré-entraîné

Accuracy Validation: 0,889

Nombre d'epochs: 44 (early stopping)

Temps d'entrainement: 21min 5s

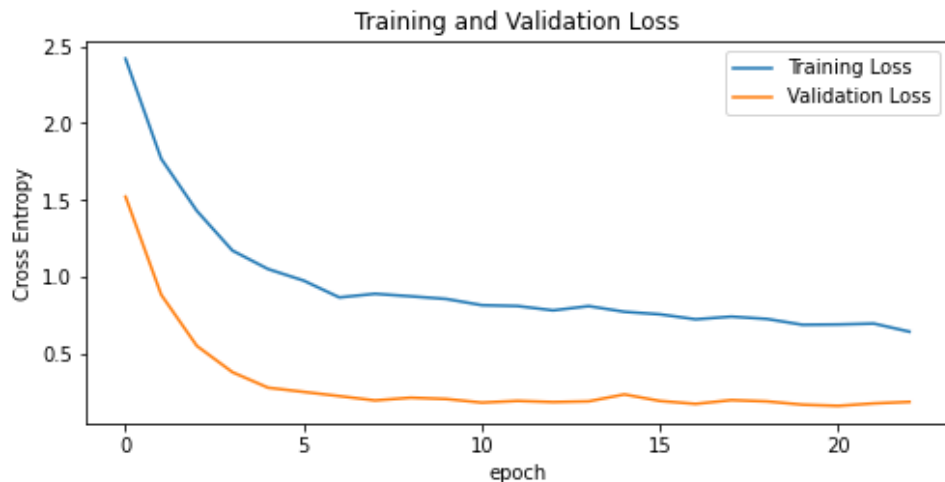
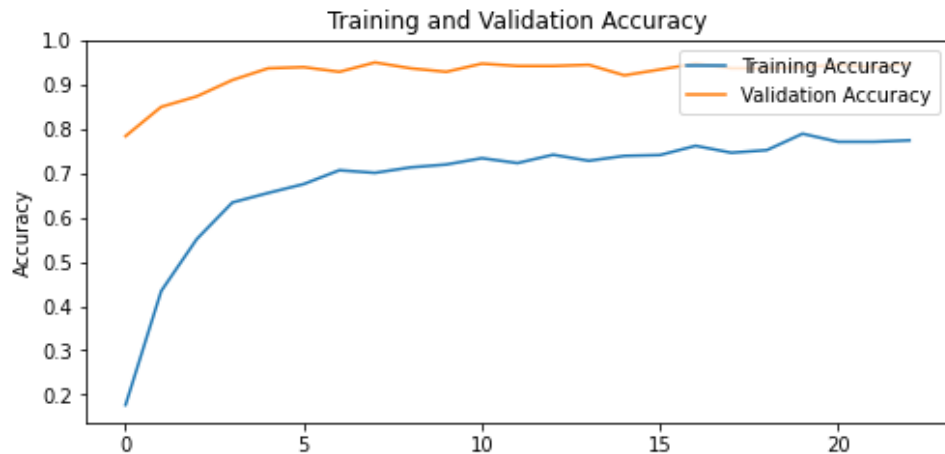


Xception pré-entraîné

Accuracy Validation: 0,949

Nombre d'epochs: 23 (early stopping)

Temps d'entrainement: 12min 21s



Comparatif des 3 modèles pré-entraînés

	ResNet50	VGG16	Xception
Accuracy sur set de validation	0,934	0,889	0,949
Nombre d'epochs: total / optimal	31 / 21	44 / 34	23 / 13
Temps d'entraînement	19min 4s	21min 5s	12min 21s

→ Xception offre une meilleure accuracy, converge plus rapidement et donc possède un temps d'entraînement plus court



4.

Modèle Final

Hyperparameters & Fine Tuning

Hyperparameters tuning avec *Hyperband*

Xception pré-entraîné

Hyperband:

- Meilleurs résultats que *RandomSearch*
- Plus rapide que *BayesianOptimization*

Hyperparamètres optimisés:

- Dropout rate: [0,3, 0,4, 0,5, 0,6]
- Learning rate: [1e-2, 1e-3, 1e-4]

Temps de recherche: 13min 55s

Meilleurs Hyperparamètres:

Dropout: 0,3

Learning rate: 1e-4

Accuracy Validation: 0,90

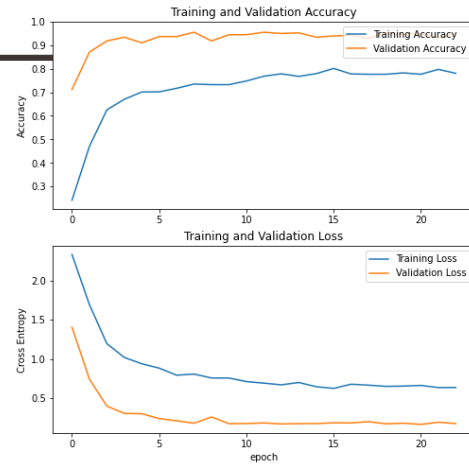
Nombre d'epochs: 2

Modèle final et Fine Tuning

Accuracy Validation: 0,955

Nombre d'epochs: 23 (early stopping)

Temps d'entrainement: 10min 33s



Fine tuning: entrainement de toutes les layers

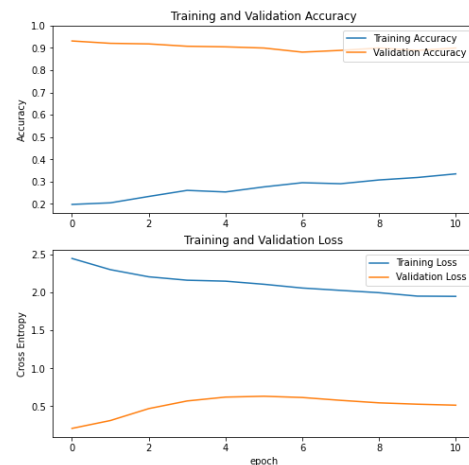
Reprise du training à la 14ème epoch

Diminution du Learning rate à $1e-5$

Accuracy Validation: 0,931

Nombre d'epochs: 11 (early stopping)

Temps d'entrainement: 8min 41s



→ Le Fine Tuning n'apporte pas de gain de performance

Pistes d'amélioration

- Obtenir plus d'images labélisées
 - Ne provenant pas d'ImageNet (les modèles ont été pré-entraînés dessus)
 - Ecart de performances faibles au regard de la taille du set de validation
 - Entraîner le modèle sur le dataset complet
 - Tuner plus d'hyperparamètres, ranges plus grandes, valeurs plus granulaires, utiliser un tuner plus performant (eg. Optimisation Bayésienne)
 - Essayer d'avantage de modèles pré-entraînés et modifier les architectures
-

Merci

fauconnier.n@gmail.com
