LE Refuge

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

Sommaire

Contexte du projet

Problématique : L'association de protection des animaux Le Refuge souhaite indexer les images des animaux qu'elle a accumulées.

Objectif : Développer un algorithme de détection automatique de la race des chiens à partir d'images.

Missions :

- Explorer les données du Stanford Dogs Dataset.
- Prétraiter les images.
- Réaliser un réseau de neurones personnel en s'inspirant des réseaux CNN existant.
- Utiliser le transfert learning en utilisant un réseau de neurones déjà entraîné.
- Mettre en place un programme de prédiction en local.

Résultat attendu :

- Un notebook contenant l'analyse et le prétraitement des images.
- Un notebook avec la création, l'entraînement et l'optimisation du modèle personnel.
- Un notebook avec l'entraînement, l'optimisation et le fine tuning du modèle déjà entraîné.
- Une API Streamlit en local afin de pouvoir réaliser les prédictions du modèle retenu.

Exploration des données .1/2

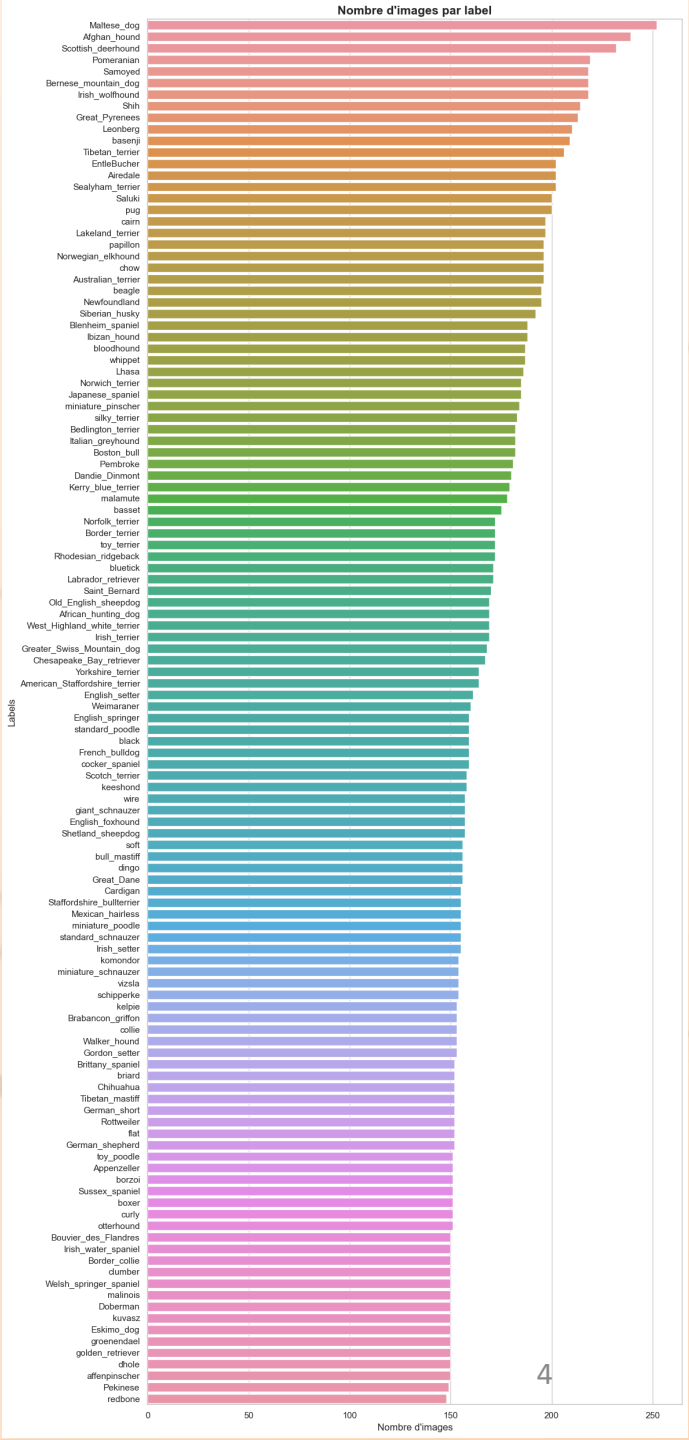
Nombre d'images par races de chiens

Le DataSet contient 120 races de chiens pour lesquelles nous avons à notre disposition plusieurs images.

En explorant le nombre d'images par race de chiens, nous pouvons voir que certaines races contiennent plus de données.

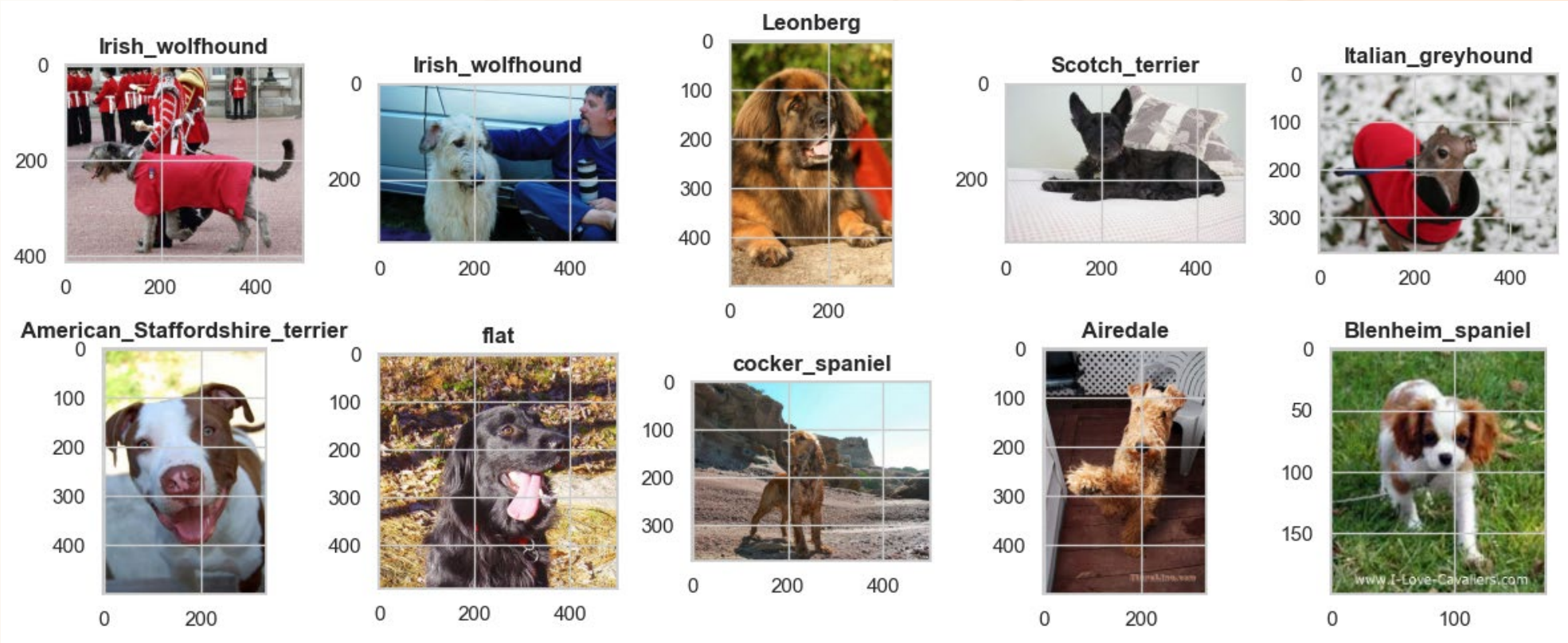
De plus, le nombre d'images est assez faible.

Nous allons donc devoir composer avec un jeu de données qui n'est pas parfait.



Exploration des données .2/2

Exemple d'images composants le DataSet



Sur cet exemple d'images, nous pouvons voir que celles-ci ne sont pas de la même dimension.

Certaines images comportent même des objets ou la présence d'humains voire d'autres chiens.

Il va falloir réaliser un prétraitement de ces images.

Prétraitement des données .1/3

Preprocessing des images

Exemple d'image

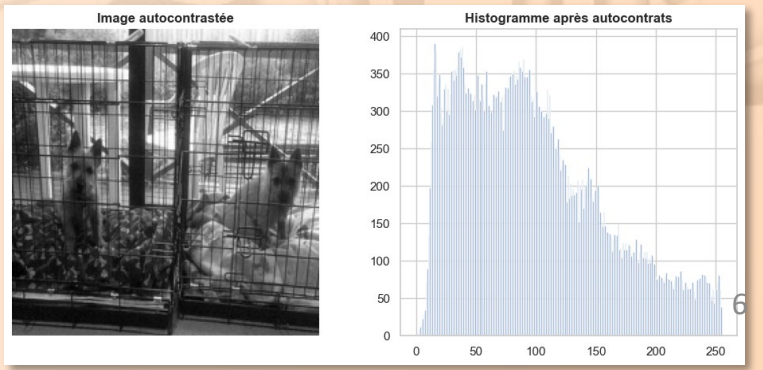
Redimensionnement de l'image :

Ici 224x224 (nous utiliserons également la taille 299x299)

Conversion en niveau de gris :

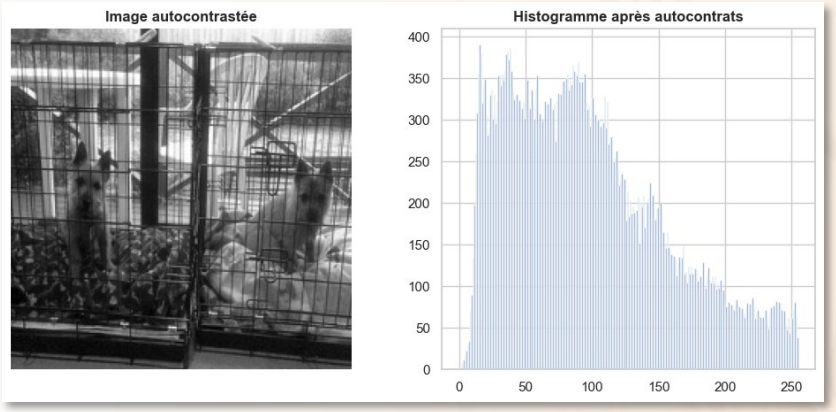
Simplifier l'image afin quelle soit représentée par un vecteur (longueur x largeur x 1)

Auto contrast sur l'image :



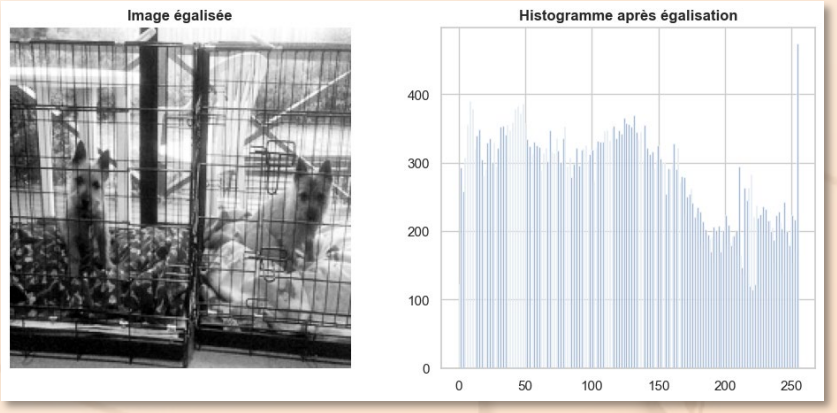
Prétraitement des données .2/3

Preprocessing des images



Suite : Image auto contrastée de la slide précédente

Egalisation de l'histogramme :



Filtrage de l'image :

Filtre Gaussien

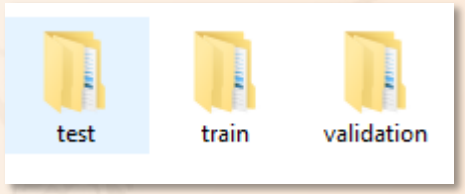
Nous allons réaliser ces différentes étapes de prétraitement des images sur la totalité de celles-ci avant d'utiliser un modèle de type réseau de neurones.

Ces étapes ont été automatisées grâce à une fonction de prétraitement des images.

Prétraitement des données .3/3

Séparation en train, validation et test set

A cette étape, l'idée est de diviser nos images prétraitées en 3 jeux de données :



Un test set.

Un train set.

Un validation set.

Le but est d'avoir nos données déjà divisées et disponibles directement dans un dossier d'images prétraitées afin de pouvoir avoir toujours la même division des données.

Modèle personnel .1/5

Description du modèle

Nous avons choisi un réseau de neurones convolutif (ou *Convolutional Neural Networks*, CNN) et nous réalisons cela sur seulement 3 races de chiens dans un premier temps.

Résumé du modèle :

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 224, 224, 32)      320

 max_pooling2d (MaxPooling2  (None, 112, 112, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 112, 112, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 56, 56, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 56, 56, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 28, 28, 128)       0
 g2D)

 conv2d_3 (Conv2D)           (None, 28, 28, 256)       295168

 max_pooling2d_3 (MaxPoolin  (None, 14, 14, 256)       0
 g2D)

 flatten (Flatten)           (None, 50176)             0

 dense (Dense)               (None, 256)               12845312

 dense_1 (Dense)             (None, 3)                 771

=================================================================
Total params: 13233923 (50.48 MB)
Trainable params: 13233923 (50.48 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Ici, plusieurs couches vont se superposer.

Une couche d'entrée :

- Taille de l'image redimensionnée : 224x224
- 32 filtres
- Fonctions d'activation Relu

Une couche de Pooling : qui va sous-échantillonner l'image (ici 112x112).

Puis une couche de convolution avec 64 filtres et toujours une fonction d'activation Relu.
Et une autre couche de Polling.
Cette étape est répétée 2 autres fois avec respectivement 128 et 256 filtres.

Une couche de Flatten permettant d'aplanir la matrice obtenue.

Une couche dense fully-connected avec 256 neurones.

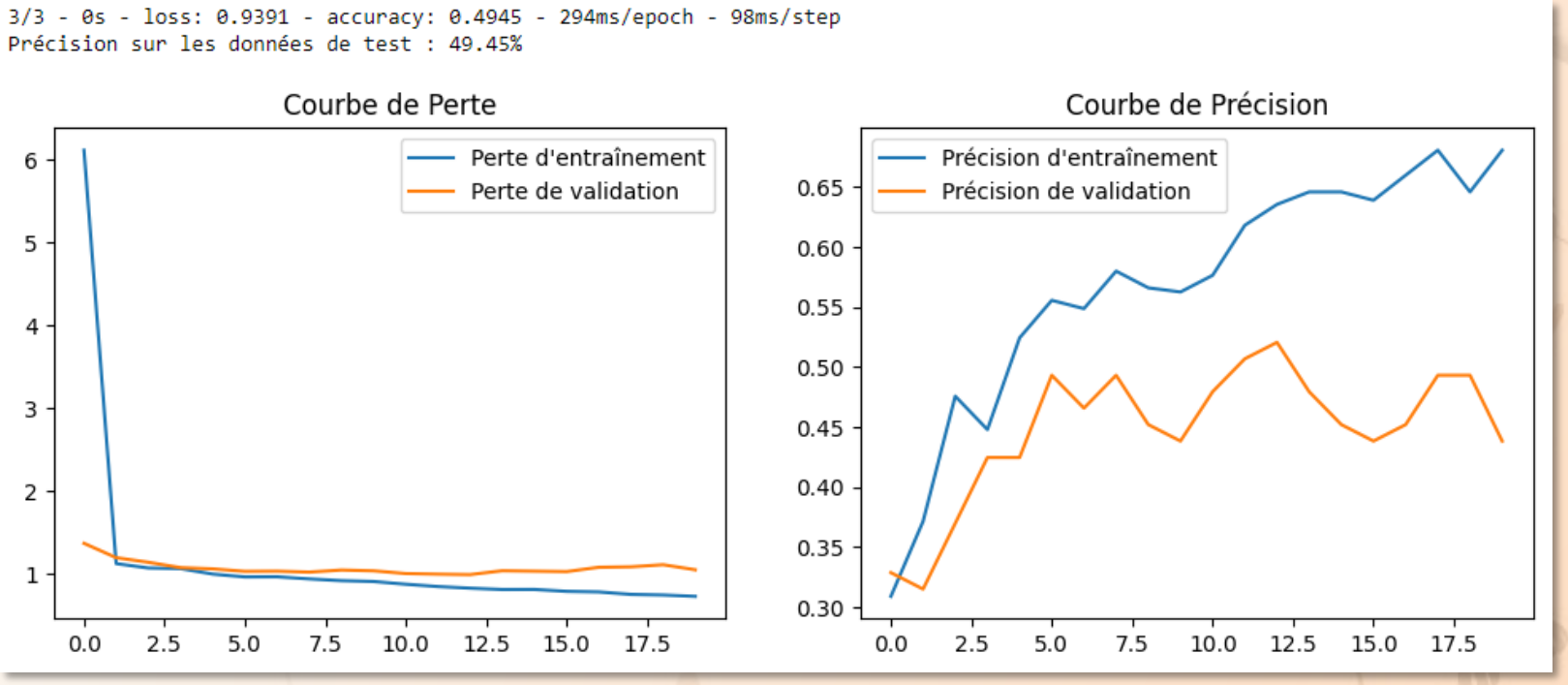
Et une couche de sortie avec une activation softmax pour prédire nos différentes classes (ici 3 pour les 3 races de chiens).

Modèle personnel .2/5

Premiers résultats sans optimisation

Nombre de classes : 3
Nombre d'epochs : 20
Batch size : 40

```
3/3 - 0s - loss: 0.9391 - accuracy: 0.4945 - 294ms/epoch - 98ms/step
Précision sur les données de test : 49.45%
```



Notre modèle non-optimisé nous permet d'obtenir une précision de 49,45% sur nos données de test pour les 3 races.

Le modèle a tendance à surapprendre sur seulement 20 epochs.

Modèle personnel .3/5

Optimisation du modèle

Nous allons chercher à optimiser un maximum d'hyperparamètres de notre réseau de neurones avec Keras Tuner. A savoir : l'activation, le nombre de filtres, le nombre de neurones de la couche dense, le taux de dropout, le learning rate et la fonction de perte.

```python
def model_builder(hp):
    """Fonction pour créer l'hypermodèle dans lequel on définit les hyperparamètres"""
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 1), padding='same'))
    model.add(MaxPooling2D(2, 2))

    # Trouver le bon nombre de filtres sur le deuxième Conv2D
    hp_filters = hp.Choice('num_filters_1', values=[32, 64], default=64)
    model.add(Conv2D(filters=hp_filters, kernel_size=3, activation='relu', padding='same'))
    model.add(MaxPooling2D(2, 2))

    # Trouver le bon nombre de filtres sur le troisième Conv2D
    hp_filters = hp.Choice('num_filters_2', values=[64, 128], default=128)
    model.add(Conv2D(filters=hp_filters, kernel_size=3, activation='relu', padding='same'))
    model.add(MaxPooling2D(2, 2))

    model.add(Flatten())

    # Trouver le bon nombre de neurones pour la couche dense
    hp_units = hp.Int('units', min_value=32, max_value=512, step=32)
    model.add(Dense(units=hp_units, activation='relu'))

    # Trouver le bon taux de Dropout
    hp_dropout = hp.Float('dropout', min_value=0.1, max_value=0.7, default=0.25, step=0.05)
    model.add(Dropout(rate=hp_dropout))

    model.add(Dense(nb_select_classes, activation='softmax'))

    # Trouver le bon learning_rate pour l'optimizer
    hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])

    # Trouver la bonne fonction de perte
    hp_loss = hp.Choice('loss_function', values=['sparse_categorical_crossentropy', 'kl_divergence'],
                        default='sparse_categorical_crossentropy')

    model.compile(optimizer=Adam(learning_rate=hp_learning_rate),
                  loss=hp_loss,
                  metrics=['accuracy'])

    return model
```

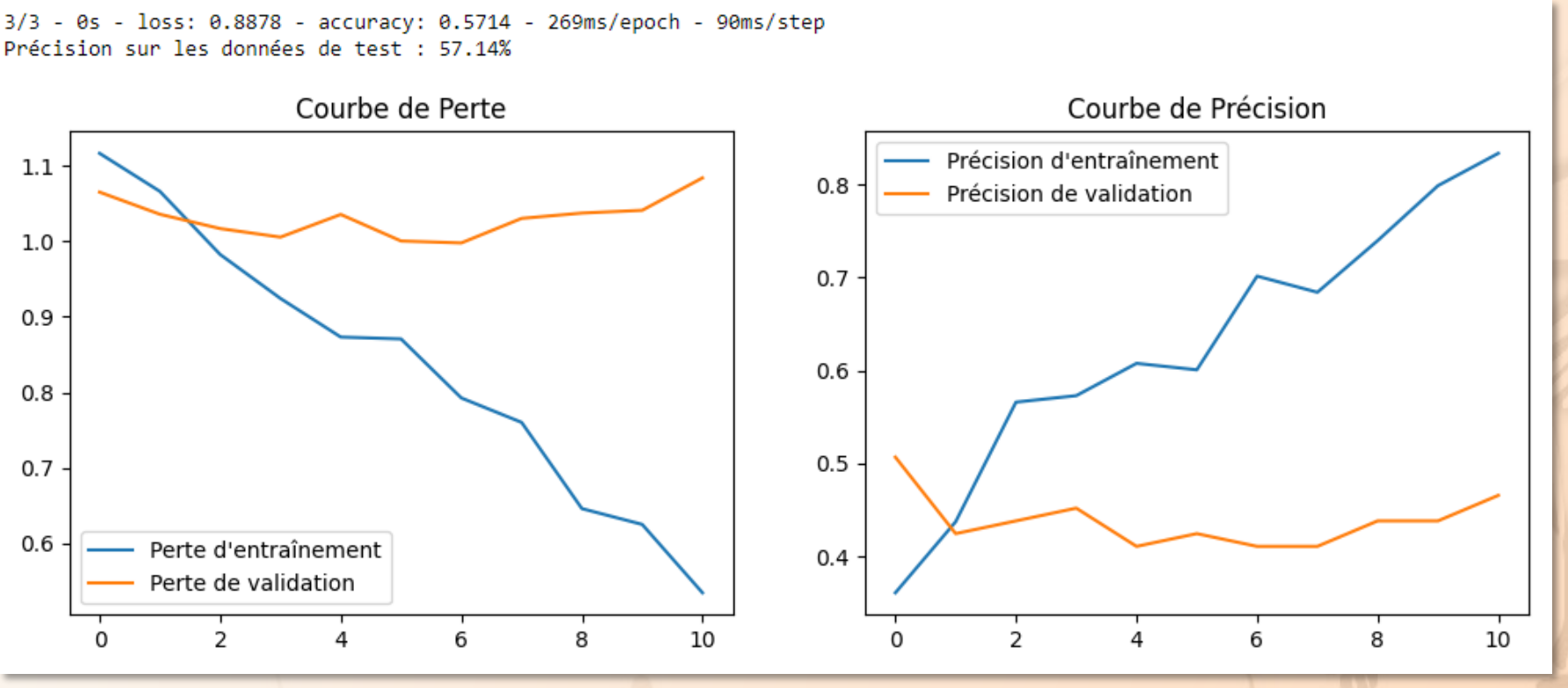
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 224, 224, 32)      320

 max_pooling2d_4 (MaxPoolin  (None, 112, 112, 32)      0
 g2D)

 conv2d_5 (Conv2D)           (None, 112, 112, 64)      18496

 max_pooling2d_5 (MaxPoolin  (None, 56, 56, 64)        0
 g2D)

 conv2d_6 (Conv2D)           (None, 56, 56, 128)       73856

 max_pooling2d_6 (MaxPoolin  (None, 28, 28, 128)       0
 g2D)

 flatten_1 (Flatten)         (None, 100352)            0

 dense_2 (Dense)             (None, 384)               38535552

 dropout (Dropout)           (None, 384)               0

 dense_3 (Dense)             (None, 3)                 1155

=================================================================
Total params: 38629379 (147.36 MB)
Trainable params: 38629379 (147.36 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Modèle personnel .4/5

Résultats après optimisation

Nombre de classes : 3
Nombre d'epochs : 11
Batch size : 40

```
3/3 - 0s - loss: 0.8878 - accuracy: 0.5714 - 269ms/epoch - 90ms/step
Précision sur les données de test : 57.14%
```



Pour optimiser notre modèle, nous avons également choisi une base de 40 epochs mais en ajoutant un EarlyStopping avec une patience à 10 ce qui nous permet de stopper avant les 40 epochs si la précision du modèle ne suit pas.

Ici le modèle a stoppé à 11 epochs et nous voyons que celui-ci est en surapprentissage.

Modèle personnel .5/5

Data Augmentation

Pour pallier le surapprentissage, nous allons utiliser la Data Augmentation car le modèle manque de données pour l'entraînement.

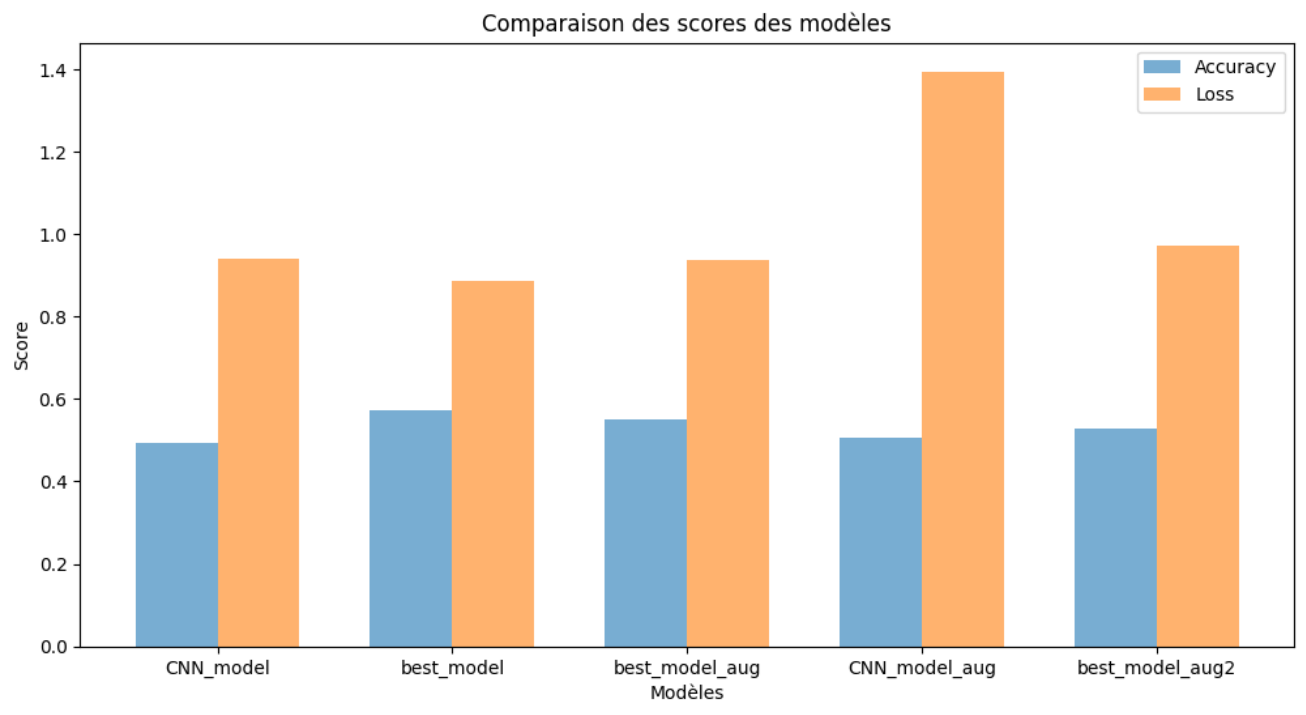
Ici, nous réaliserons des rotations et étirements de l'image comme ci-dessous :

Résultat de la data augmentation suite à rotation :

Résultat de la data augmentation suite à étirement de l'image :

Choix du modèle

Comparaison des résultats



	Model	Accuracy	Loss	Temps d'entraînement du modèle
1	best_model	0.571429	0.887784	41.051292
2	best_model_aug	0.549451	0.938868	49.609229
4	best_model_aug2	0.527473	0.970796	56.935886
3	CNN_model_aug	0.505495	1.395484	101.563106
0	CNN_model	0.494505	0.939130	68.554517

Résumé :
CNN_model = modèle de base non optimisé

best_model = model de base optimisé

best_model_aug = model optimisé avec Data Augmentation

CNN_model_aug = modèle avec une couche de convolution supplémentaire

best_model_aug2 = modèle CNN_model_aug optimisé

Ici, nous constatons que la précision est meilleure pour 'best_model', mais ce modèle présentait une tendance au surapprentissage (overfitting).

En augmentant les données, nous avons réussi à atténuer ce phénomène, rendant le modèle plus stable avec l'ajout d'une couche de convolution supplémentaire.

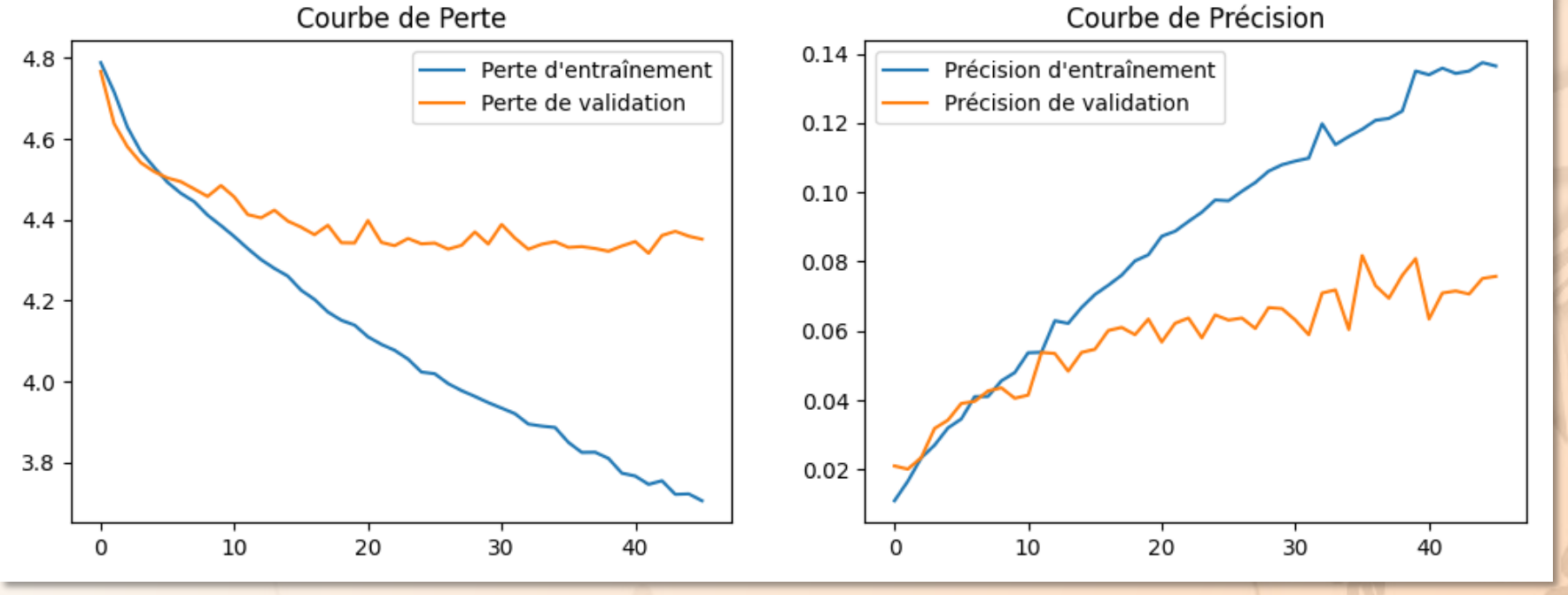
Nous allons donc opter pour le modèle 'best_model_aug2', qui affiche une précision de 0,527 sur un échantillon de 3 races de chiens, avant de le tester sur l'ensemble des 120 races du dataset.

Utilisation du modèle

Essai du modèle sur les 120 races de chiens

Nombre de classes : 120
Nombre d'epochs : 46
Batch size : 150

```
Found 4162 images belonging to 120 classes.
28/28 - 11s - loss: 4.5959 - accuracy: 0.0560 - 11s/epoch - 403ms/step
Précision sur les données de test : 5.60%
```



Une fois le modèle entraîné avec les 120 classes, celui-ci ne donne pas de résultats concluants.

Nous allons donc utiliser une autre technique afin de pouvoir prédire les races de chiens en fonction des images.

Modèles pré-entraînés .1/3

Description des modèles

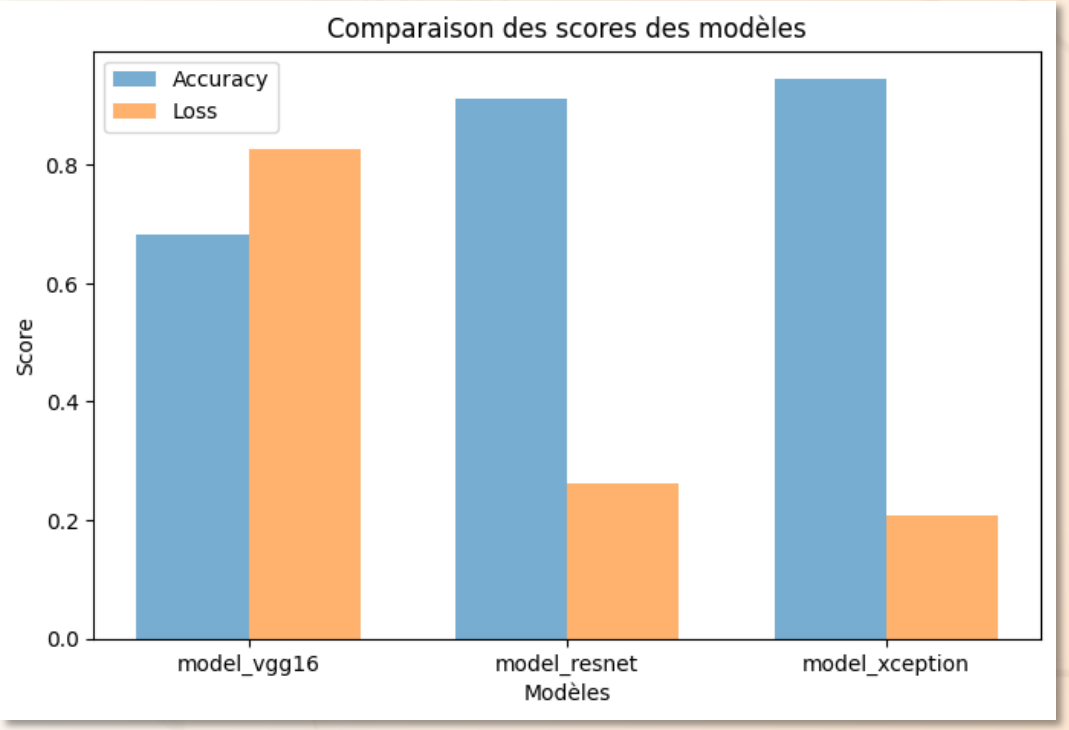
Nous allons utiliser des modèles pré-entraînés afin de réaliser du transfert learning.

L'idée est que nous disposons de modèles plus complexes avec des couches déjà pré-entraînées et nous allons donc modifier la couche de sortie de ces modèles afin de créer un nouveau classifier qui sera propre à nos classes.

Modèles pré-entraînés .2/3

Comparaison des résultats

Nombre de classes : 3
Nombre d'epochs : 40 (avec EarlyStopping)
Batch size : 40



	Model	Accuracy	Loss	Temps d'entraînement du modèle
2	model_xception	0.945055	0.207006	198.450920
1	model_resnet	0.912088	0.262377	109.711118
0	model_vgg16	0.681319	0.825224	333.340772

Le modèle Xception est meilleur.

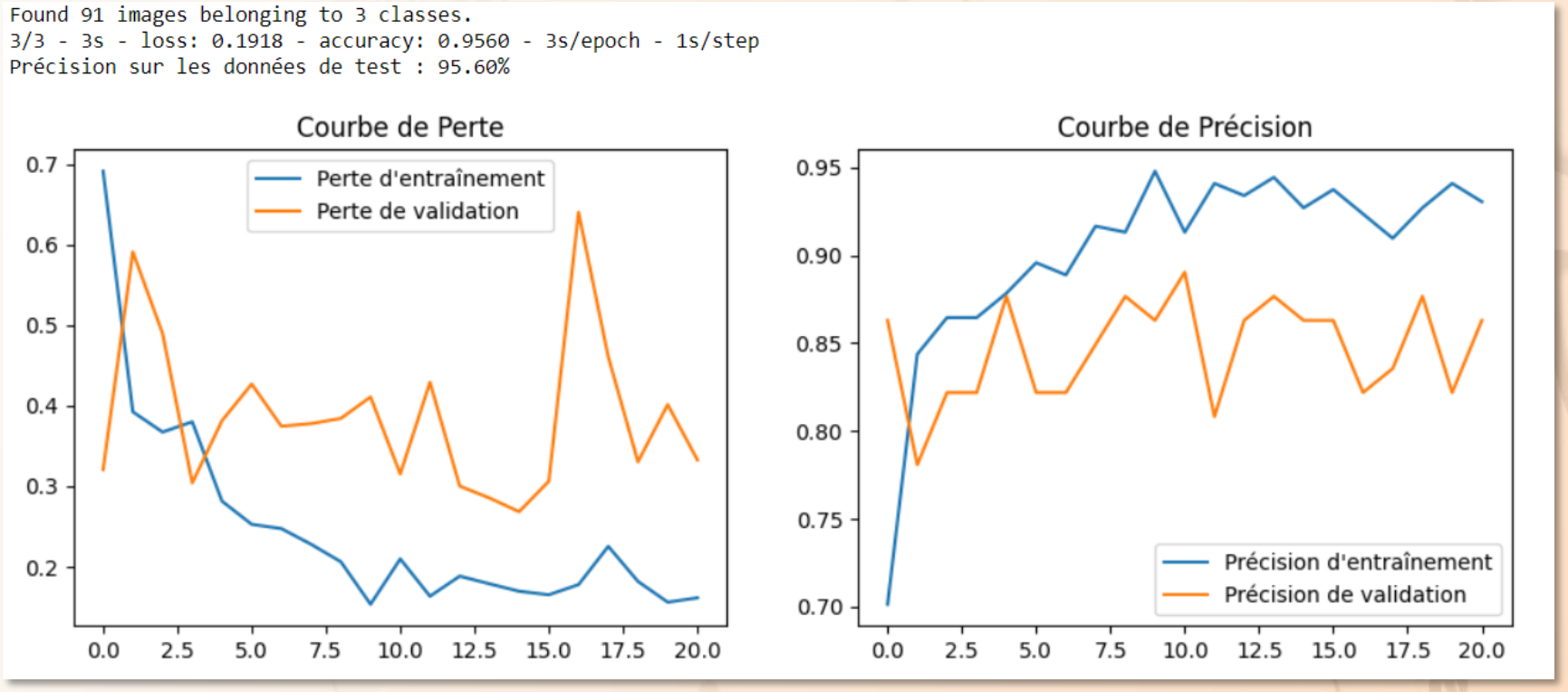
Nous allons donc optimiser les paramètres de celui-ci et tenter d'entraîner le modèle sur plus de races de chiens.

Modèles pré-entraînés .3/3

Optimisation du modèle

Nombre de classes : 3
Nombre d'epochs : 21
Batch size : 40

```
Found 91 images belonging to 3 classes.
3/3 - 3s - loss: 0.1918 - accuracy: 0.9560 - 3s/epoch - 1s/step
Précision sur les données de test : 95.60%
```



Sur le même principe que le modèle personnel, nous avons utilisé keras-tuner afin d'optimiser les hyperparamètres.

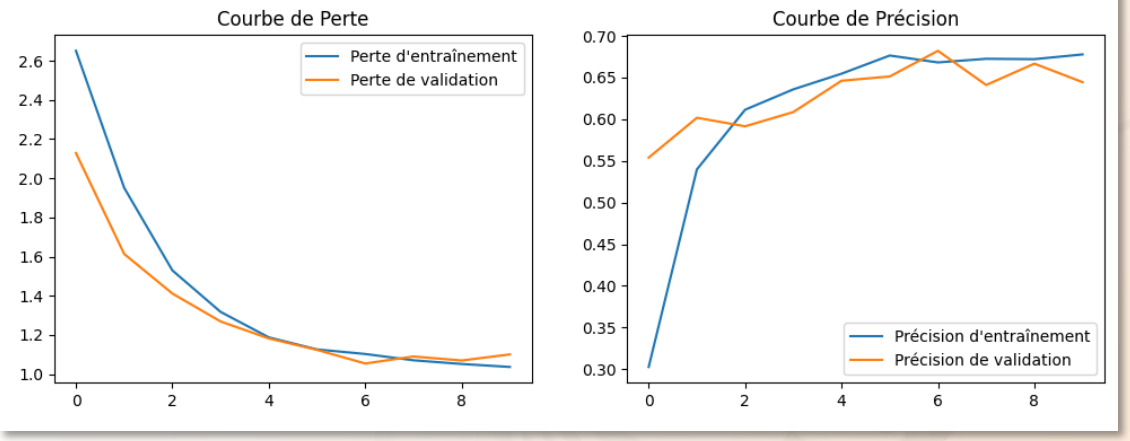
Nous obtenons une légère hausse de l'accuracy, nous garderons donc les hyperparamètres de ce modèle pour essayer de prédire plus de 3 races de chiens.

Choix du modèle

Nombre de races de chiens

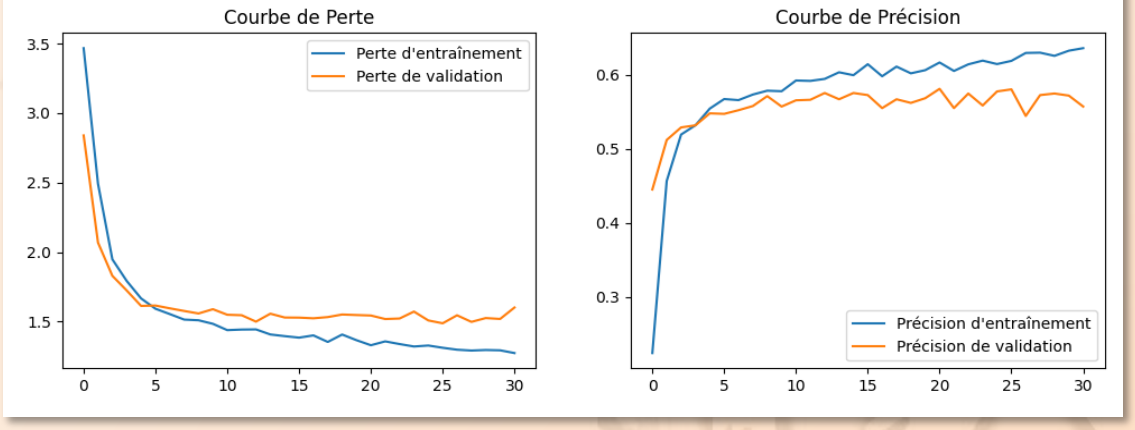
20 races de chiens:

```
5/5 - 46s - loss: 0.6246 - accuracy: 0.7846 - 46s/epoch - 9s/step
Précision sur les données de test : 78.46%
```



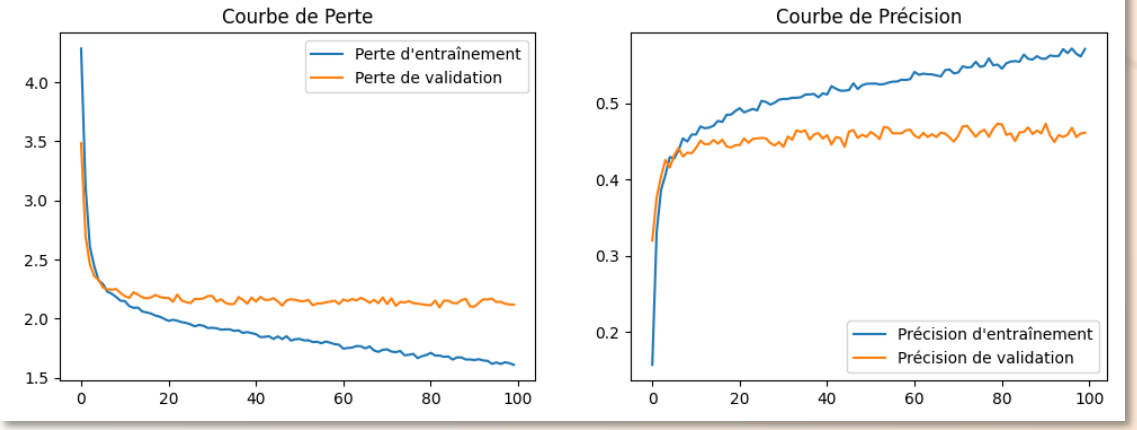
50 races de chiens:

```
Found 1781 images belonging to 50 classes.
12/12 - 57s - loss: 0.8535 - accuracy: 0.7339 - 57s/epoch - 5s/step
Précision sur les données de test : 73.39%
```



120 races de chiens:

```
Found 4162 images belonging to 120 classes.
28/28 - 128s - loss: 1.3417 - accuracy: 0.6307 - 128s/epoch - 5s/step
Précision sur les données de test : 63.07%
```



Nous avons testé le modèle Xception optimisé sur 20, 50 et 120 races de chiens.

Le but est d'obtenir un modèle qui va nous permettre de prédire de manière efficace un maximum de classes.

Toutefois, pour avoir un modèle cohérent nous devons prendre celui qui fera le moins d'erreur.

Ici, c'est le modèle qui prédit 20 races de chiens avec une accuracy de 78.46%.

Fine tuning .1/2

Choix des paramètres

L'idée ici est d'essayer d'améliorer le modèle en réentraînant certaines couches du modèle pré-entraîné mais avec nos données.

Choix du réseau de neurones qui prédit 20 races de chiens

Fine tuning

Input du modèle

Couches pré-entraînées

Couches pré-entraînées mais que l'on réentraîne

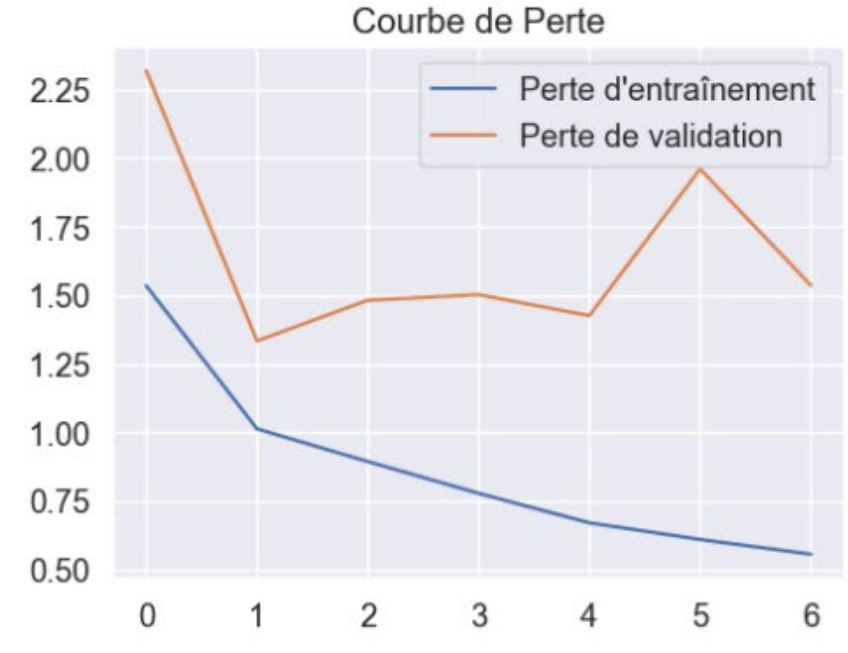
Couches de sorties + Classifier

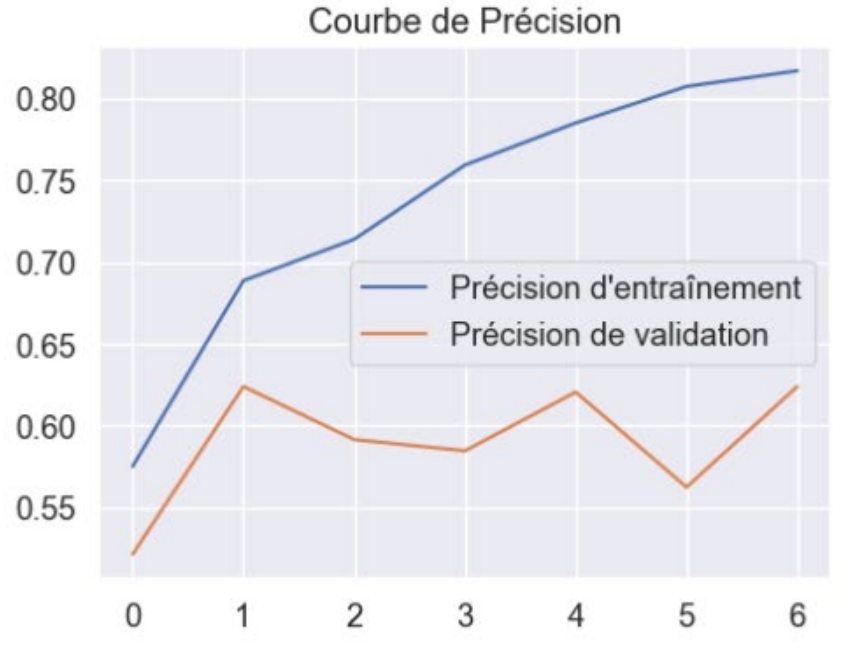
On réentraîne notre modèle sur les 15 dernières couches.

Fine tuning .2/2

Résultats

```
5/5 - 45s - loss: 1.0555 - accuracy: 0.7682 - 45s/epoch - 9s/step
Précision sur les données de test : 76.82%
```





Le problème est que le modèle sur-apprend et devient moins bon.

Nous garderons donc le modèle optimisé sans le fine tuning pour notre API.

Démonstration API

Streamlit

Afin de pouvoir avoir un outil intuitif pour réaliser les prédictions, nous avons choisis de réaliser une API avec en backend une API Flask et en frontend une interface Streamlit.

Cette application est disponible en local.