

Projet 6

Classez des images à l'aide
d'algorithme de Deep learning

Plan de présentation

1. Présentation de problématique
2. Environnement de travail
3. Mise en forme de données visuelles
4. Modélisation
 - A. Entraînement de mon propre réseau de neurones convolutif (CNN)
 - B. Transfer learning
5. Comparaison des deux méthodes
6. Prédictions
7. Conclusion

1. PRÉSENTATION DE PROBLÉMATIQUE

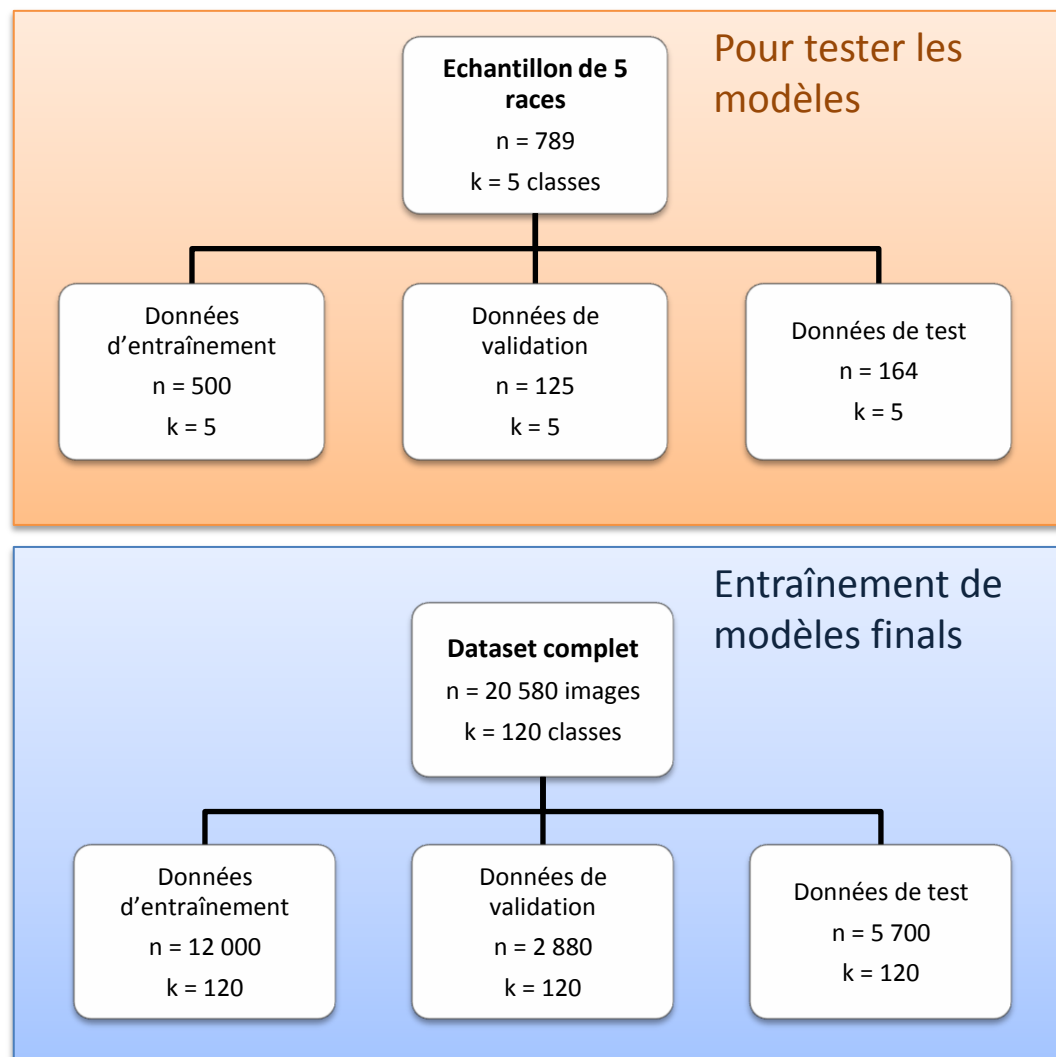
1. Présentation de problématique

- Notre partenaire, une association de protection d'animaux, dispose d'une photothèque de chiens de différents races qui est devenue assez volumineuse pour un étiquetage humaine
- Notre mission consiste à développer un algorithme de Deep Learning qui aidera l'association à classer automatiquement les photos de chiens selon leur race
- Dans un premier temps, nous allons développer notre propre algorithme de Deep Learning (CNN). Ensuite, nous allons comparer ses performances à un modèle créé à partir d'un algorithme déjà existant à l'aide de Transfer Learning
- Etant donné que l'entraînement d'un algorithme pour classifier des données visuelles est assez gourmand en ressources, nous allons tester 3 environnements de travail différents et comparer leur performance

1. Présentation de problématique

- Nous disposons d'une photothèque de chiens, Stanford Dogs Dataset qui contient :
 - 120 catégories (races de chien)
 - 20 580 images

- Data flow :



2. ENVIRONNEMENT DE TRAVAIL

2. Environnement de travail

1. PC local: CPU avec 32Go de RAM

- **Avantages :**
 - Environnement déjà paramétré
- **Inconvénients :**
 - Rapidité de calcul



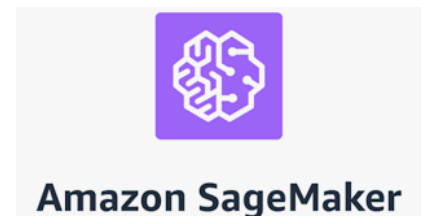
2. Google colab: GPU avec ~13Go de RAM

- **Avantages :**
 - Gratuité de mise à disposition
- **Inconvénients :**
 - Taille de RAM relativement limitée
 - Espace de stockage insuffisant
 - Le serveur se déconnecte après 30 min d'inactivité => pas approprié pour des longues exécutions



3. AWS SageMaker: CPU avec 61Go + GPU avec 16Go de RAM

- **Avantages :**
 - Nous pouvons choisir le matériel selon nos besoins => choix d'une instance de calcul accéléré
- **Inconvénients :**
 - Service payant
 - Paramétrage d'environnement de travail à prendre en main

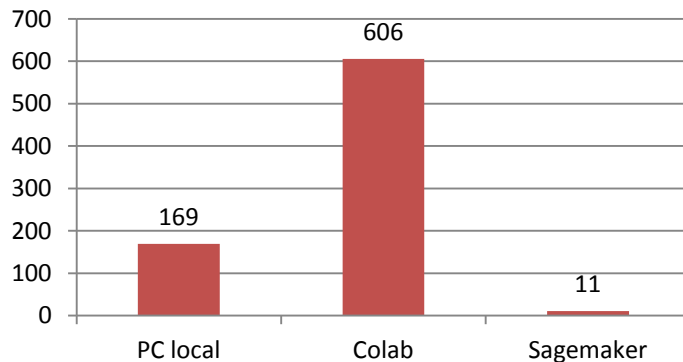


2. Environnement de travail

Comparaison de performances :

- Entraînement d'un CNN (réseau de neurones convolutif)
 - Echantillon de 5 races
 - 10 epochs
 - Modèle de base avec 1,4 million de paramètres d'entraînement

Temps d'exécution (en sec)



Répartition de travail :

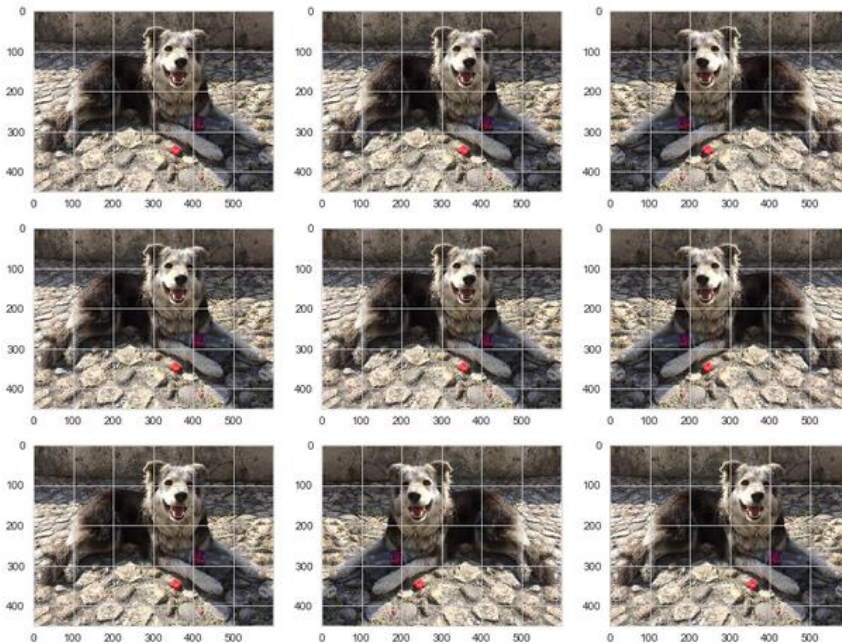
1. **Colab** : Tests sur l'échantillon : mon CNN + Transfer learning
2. **PC local** : mon CNN sur les données complètes
3. **Sagemaker** : Exécuter les 2 modèles finales sur les données complètes + augmentation de nombre d'epochs

3. MISE EN FORME DE DONNÉES VISUELLES

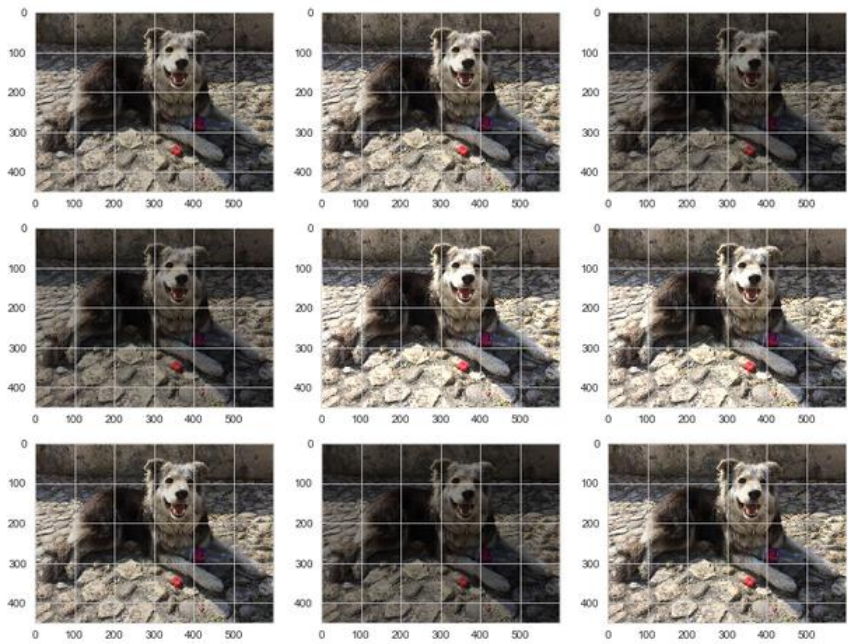
3. Mise en forme de données visuelles

- Augmentation de données visuelles :
 - Modification aléatoire de certaines images dans le dataset d'entraînement
 - Méthode qui vise à généraliser les données et réduire le risque de surapprentissage
- Exemples d'augmentation :
 - Shift horizontale / verticale -> décaler le centre d'image (de % de taille d'image définit)
 - Flip horizontal -> inverser l'image (effet de miroir)
 - Rotation -> rotation d'image de l'angle défini
 - Brightness -> modification de luminosité d'image

Horizontal flip



Brightness



3. Mise en forme de données visuelles

Application:

- Méthode ImageDataGenerator intégrée dans le package Keras
- Nous créons un générateur de données qui (selon les options) s'occupe de :
 - Créer des batchs de taille prédéfini, où les photos sont tirées dans l'ordre aléatoire
 - Normaliser les données
 - Redimensionner les images
 - Augmenter les images (données d'entraînement)
 - Loader les batch un par un dans le modèle
- Avantage de générateur par batch: Nous ne sommes pas obligé de charger tout le dataset dans la mémoire
- Avantage de la méthode : Feature engineering est fait en quelques lignes de code



3. Mise en forme de données visuelles

Exemple de code – données d'entraînement:

```
Entrée [28]: batch_size = 20
             IMG_HEIGHT = 224
             IMG_WIDTH = 224
```

```
Entrée [29]: # Fit the generator including data augmentation
             train_generator = ImageDataGenerator(rescale=1./255,
                                                  width_shift_range = 0.2,
                                                  height_shift_range = 0.2,
                                                  horizontal_flip=True,
                                                  rotation_range=20,
                                                  brightness_range= [0.5, 1.2])
```

```
Entrée [30]: train_data_gen = train_generator.flow_from_directory(directory=str(train_data_dir),
                                                                seed=23,
                                                                batch_size=batch_size,
                                                                shuffle=True,
                                                                target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                                classes = list(CLASS_NAMES),
                                                                class_mode="sparse")
```

Found 12000 images belonging to 120 classes.

Exemple de code – données de validation:

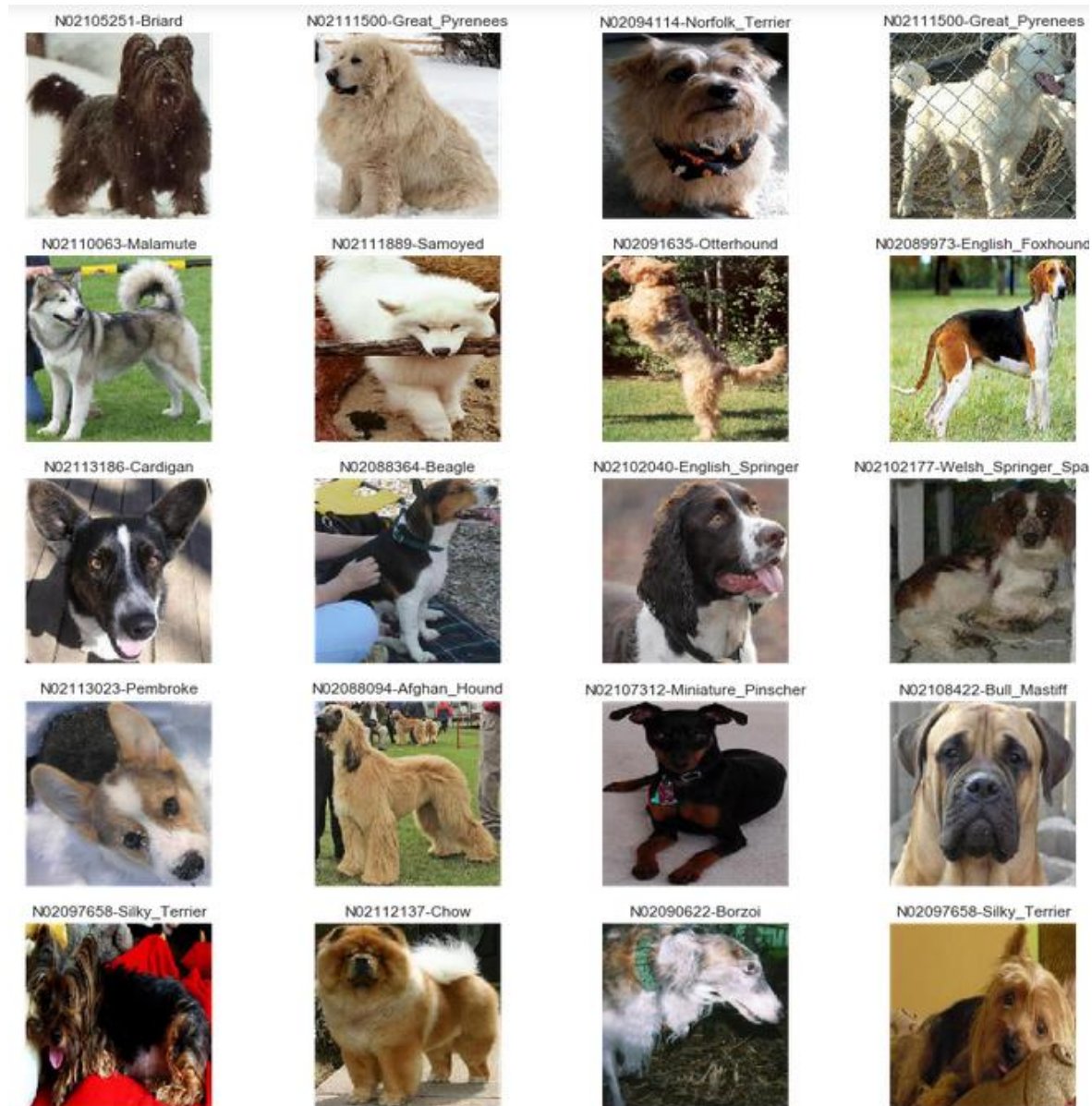
```
Entrée [31]: # The 1./255 is to convert from uint8 to float32 in range [0,1].
             test_generator = ImageDataGenerator(rescale=1./255)
```

```
Entrée [32]: valid_data_gen = test_generator.flow_from_directory(directory=str(valid_data_dir),
                                                                seed=23,
                                                                batch_size=batch_size,
                                                                shuffle=True,
                                                                target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                                classes = list(CLASS_NAMES),
                                                                class_mode="sparse"
                                                                )
```

Found 2880 images belonging to 120 classes.

3. Mise en forme de données visuelles

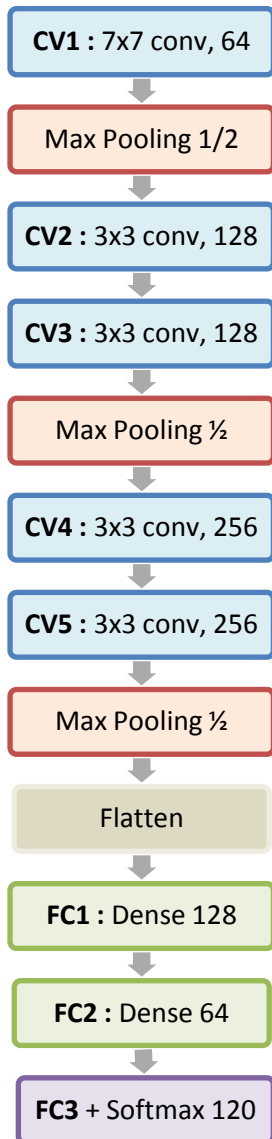
Exemple d'un batch de 20 images



4. MODÉLISATION

A – Entraînement de mon propre CNN

Modèle de base - Architecture



```
my_CNN1 = keras.models.Sequential([
    keras.layers.Conv2D(64, 7, activation="relu", padding="same", strides=2, input_shape=[224, 224, 3]),
    keras.layers.MaxPooling2D(2),
    keras.layers.Conv2D(128, 3, activation="relu", padding="same", strides=2),
    keras.layers.Conv2D(128, 3, activation="relu", padding="same", strides=2),
    keras.layers.MaxPooling2D(2),
    keras.layers.Conv2D(256, 3, activation="relu", padding="same", strides=1),
    keras.layers.Conv2D(256, 3, activation="relu", padding="same", strides=1),
    keras.layers.MaxPooling2D(2),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dense(64, activation="relu"),
    keras.layers.Dense(120, activation="softmax")
])
my_CNN1.summary()
```

- **Couche convolutionnelle (exemple CV1):**

- 64 filtres de taille 7x7 => fenêtres qui parcourent la photo à la recherche de features
- strides : taille de pas de déplacement de filtres
- padding = same : ajouter une bordure noire pour ne pas réduire la taille d'image en sortie
- Activation par une couche ReLU ($\max(0, x)$)

- **Couche de pooling :**

- Calcule le max d'une fenêtre 2*2 pixels => réduction de nombre de pixels en sortie

- **Couche flatten:**

- Réduction de données en matrice 3D dans un vecteur

- **Couches Fully connected:**

- activation ReLU
- Dernière couche activation par softmax avec 120 classes à prédire

- **Nombre de paramètres d'entraînement : 1.4 millions**

A – Entraînement de mon propre CNN

Recherche de paramètres optimales

1. Taille de batch

- Est-ce que la taille de batch influence sur la vitesse d'exécution et la précision ?

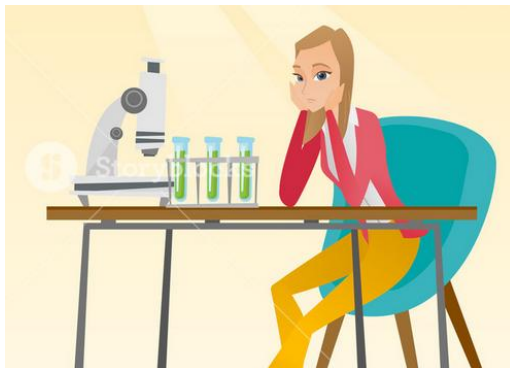
2. Changement de paramètre strides

- Peut-on gagner en précision ?

3. Modification de la structure de modèle

- Réduction de nombre de paramètres pour gagner en vitesse
- Ajoute de couche dropout

4. Tester des learning rates évolutifs avec Adam et power scheduling

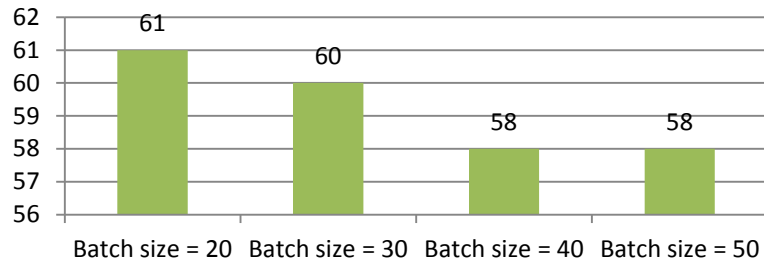


A – Entraînement de mon propre CNN

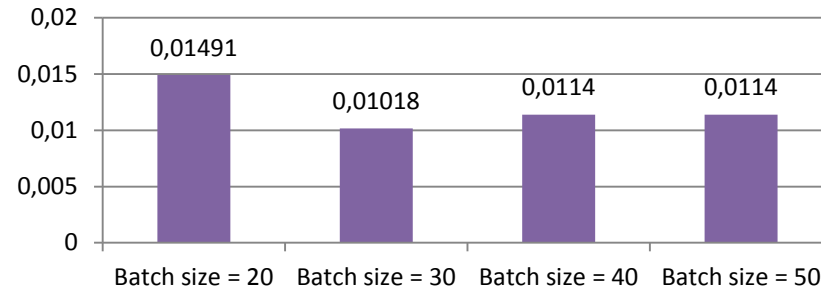
Optimiser la taille de batch

- Testé sur les données complètes sur le PC local
- Exécution en 10 epochs

Temps d'exécution (min)

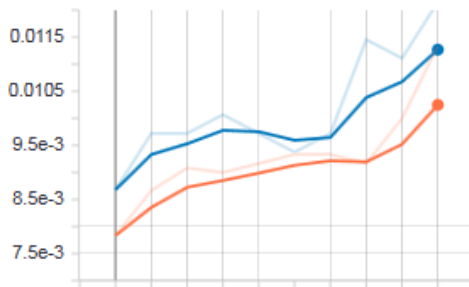


Précision

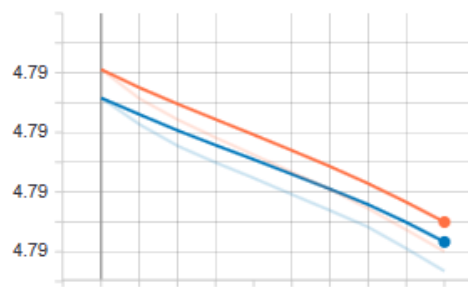


Courbes d'apprentissage – batch size = 20

epoch_accuracy



epoch_loss



Name	Smoothed	Value	Step	Time	Relative
my_CNN1A_run_2019_12_30-09_37_34\train	0.01025	0.01133	9	Mon Dec 30, 10:38:45	54m 22s
my_CNN1A_run_2019_12_30-09_37_34\validation	0.01127	0.01215	9	Mon Dec 30, 10:38:45	54m 22s

Conclusion :

- La taille de batch n'a pas beaucoup d'influence sur le temps d'exécution, ni sur la précision
- Le temps d'exécution est assez important
- La précision est très basse
- 10 epochs n'est pas assez pour voir l'évolution de modèle, car la fonction de perte a encore une tendance à diminuer

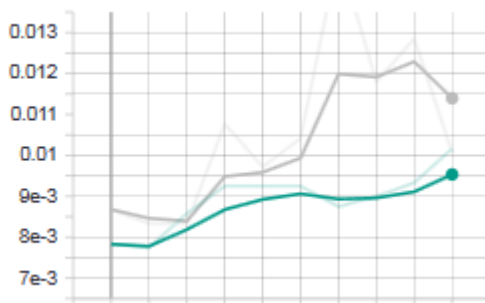
A – Entraînement de mon propre CNN

Changement de paramètre strides

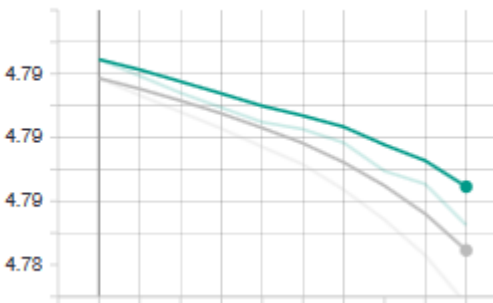
```
my_CNN3 = keras.models.Sequential([
    keras.layers.Conv2D(64, 7, activation="relu", padding="same", strides=2, input_shape=[224, 224, 3]),
    keras.layers.MaxPooling2D(2),
    keras.layers.Conv2D(128, 3, activation="relu", padding="same", strides=1), # Changed from 2 -> 1
    keras.layers.Conv2D(128, 3, activation="relu", padding="same", strides=1),
    keras.layers.MaxPooling2D(2),
    keras.layers.Conv2D(256, 3, activation="relu", padding="same", strides=1),
    keras.layers.Conv2D(256, 3, activation="relu", padding="same", strides=1),
    keras.layers.MaxPooling2D(2),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dense(64, activation="relu"),
    keras.layers.Dense(120, activation="softmax")
])
my_CNN3.summary()
```



- Il s'agit de paramètre qui indique la taille de pas de déplacement de la fenêtre (ici 3x3)
- Le changement de paramètres entraîne le changement de :
 - Paramètres d'entraînement: 1,4 -> 7,5 millions
 - Temps d'exécution : 1h 1min -> 3h 8min
 - Précision : 1,491% -> 1,860 %

epoch_accuracy



epoch_loss



Name	Smoothed	Value	Step	Time	Relative
 my_CNN3_run_2019_12_31-13_34_44\train	9.5359e-3	0.01017	9	Tue Dec 31, 16:45:40	2h 49m 55s
 my_CNN3_run_2019_12_31-13_34_44\validation	0.0114	0.01007	9	Tue Dec 31, 16:45:40	2h 49m 55s

A – Entraînement de mon propre CNN

Diminuer le nombre de paramètres – nouvelle architecture

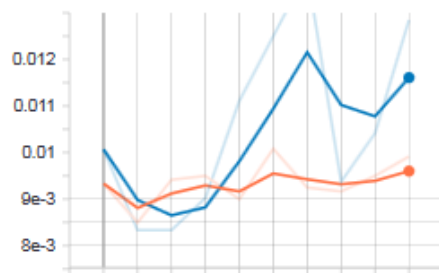
```
my_CNN4 = keras.models.Sequential([
    keras.layers.Conv2D(64, 7, activation="relu", padding="same", strides=2, input_shape=[224, 224, 3]),
    keras.layers.MaxPooling2D(2),
    keras.layers.Conv2D(128, 3, activation="relu", padding="same", strides=1),
    keras.layers.Conv2D(128, 3, activation="relu", padding="same", strides=1),
    keras.layers.MaxPooling2D(2),
    keras.layers.Conv2D(256, 3, activation="relu", padding="same", strides=1),
    keras.layers.Conv2D(256, 3, activation="relu", padding="same", strides=1),
    keras.layers.MaxPooling2D(2),
    keras.layers.Flatten(),
    #keras.layers.Dense(256, activation="relu"),
    #keras.layers.Dense(128, activation="relu"),
    keras.layers.Dense(64, activation="relu"),
    keras.layers.Dropout(0.5), # New layer added
    keras.layers.Dense(120, activation="softmax")
])
```

1. Suppression de 2 couches fully-connected
2. Ajout d'une couche Dropout:
 - Active seulement 50% de neurones aléatoires => augmentation de vitesse de calcul et évite surapprentissage

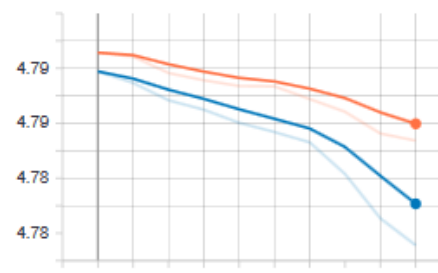
Conclusion:

- Paramètres : 7,5 -> 4,3 million
- Temps d'exécution : 3h 8min -> 3h 7min
- Précision : 1,860 -> 2,158 %

epoch_accuracy



epoch_loss



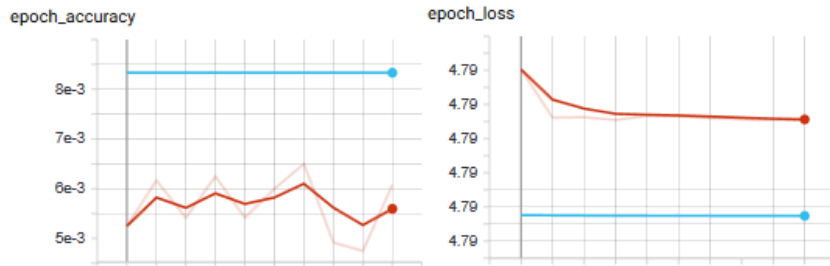
Name	Smoothed	Value	Step	Time	Relative
my_CNN4_run_2019_12_31-16_57_21\train	9.6041e-3	9.9167e-3	9	Tue Dec 31, 20:04:56	2h 48m 38s
my_CNN4_run_2019_12_31-16_57_21\validation	0.01161	0.01285	9	Tue Dec 31, 20:04:56	2h 48m 38s

A – Entraînement de mon propre CNN

Learning rate évolutif

1. Algorithme Adam (Adaptive moment estimation)

- Alternative à la méthode de descente de gradient stochastique
- Combinaison d'Adaptive Gradient Algorithm (AdaGrad) et Root Mean Square Propagation (RMSProp)



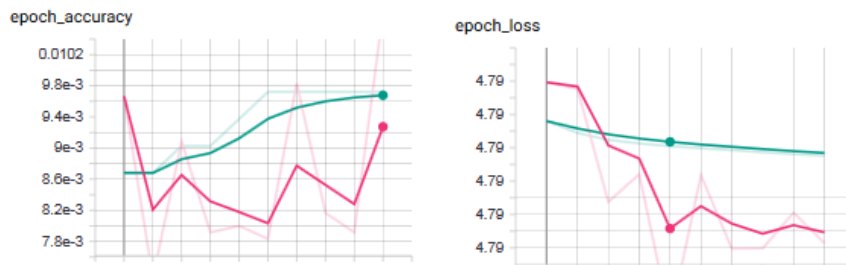
Test accuracy = 0.01649

Temps d'exécution = 3h 4min

Name	Smoothed	Value	Step	Time	Relative
my_CNN5_run_2019_12_31-20_06_51\train	5.5962e-3	6.0833e-3	9	Tue Dec 31, 23:11:00	2h 45m 40s
my_CNN5_run_2019_12_31-20_06_51\validation	8.3333e-3	8.3333e-3	9	Tue Dec 31, 23:11:00	2h 45m 40s

2. Power scheduling

- Learning rate est défini en fonction de nombre d'itérations t
- Après un certains nombre de pas (que nous définissons comme paramètre), le learning rate est divisé par une constante qui augmente avec le nombre de pas



Test accuracy = 0.00930

Temps d'exécution = 3h 7min

Name	Smoothed	Value	Step	Time	Relative
my_CNN6_run_2019_12_31-23_12_53\train	9.2743e-3	0.01075	9	Wed Jan 1, 02:19:58	2h 48m 23s
my_CNN6_run_2019_12_31-23_12_53\validation	9.6792e-3	9.7222e-3	9	Wed Jan 1, 02:19:58	2h 48m 23s

A – Entraînement de mon propre CNN

Conclusion

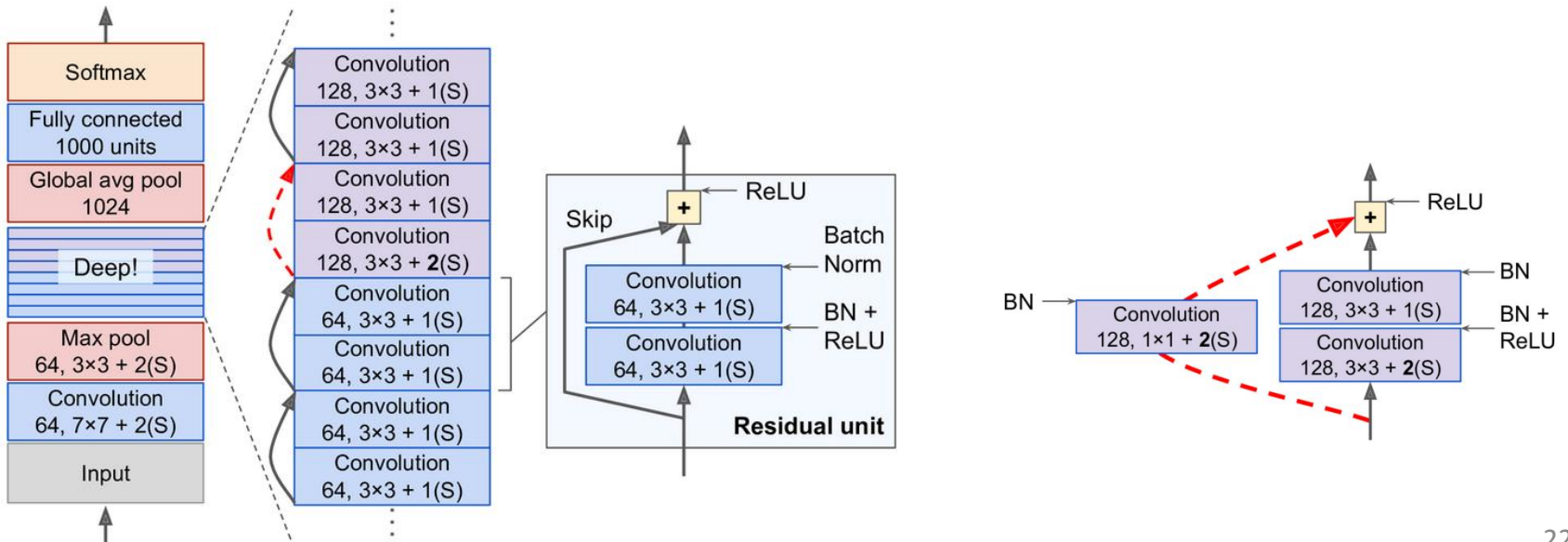
- L'exécution de modèle en 10 epochs n'est pas suffisant, mais dure environ 3h sur le PC local
 - **Solution** : Pour comparer la méthode avec le transfer learning, le modèle va être exécuté à nouveau en 40 epochs à l'aide de Sagemaker
- Le meilleur modèle testé nous donne une précision de 2.158 %
 - D'après la revue d'articles sur le dataset, il n'est pas surprenant de ne pas avoir une grande précision avec un modèle construit à partir de scratch
 - Dataset ne contient pas un nombre suffisant de photos pour la complexité de tâche
 - **Solution** : utiliser un modèle déjà entraîné, tester l'extraction de features et fine-tuning partiel

B - Transfer Learning

Utilisation de deux modèles :

1. ResNet50 (2015)
 2. Xception (2016)
- Remplacement de la dernière couche fully connected par une couche activée par softmax avec le nombre de paramètres qui corresponde à notre problème
 - La photothèque dont nous disposons étant relativement petite, nous allons tester 2 méthodes de transfer learning :
 1. Feature extraction :
 - Utilisation de poids d'imagenet
 2. Fine tuning partiel :
 - Utilisation de poids d'imagenet seulement dans x premières couches, entraînement de reste du réseau

Exemple d'architecture de ResNet :



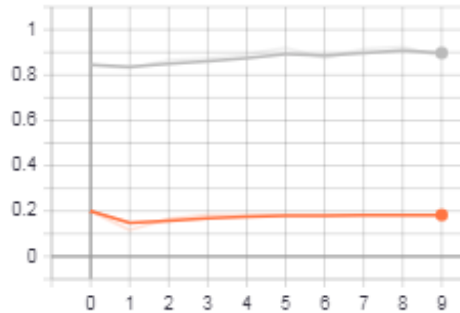
Transfer Learning – Tests sur échantillon

Extraction de features

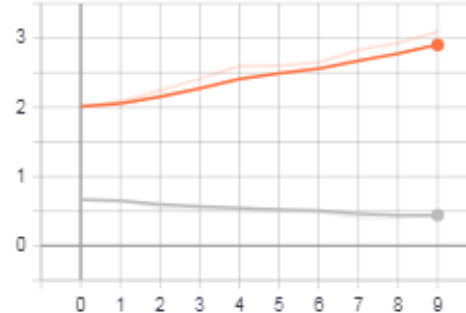
ResNet50

Batch size = 30

epoch_accuracy



epoch_loss



- Temps d'exécution : 1h 2min
- Précision : 26.83 %

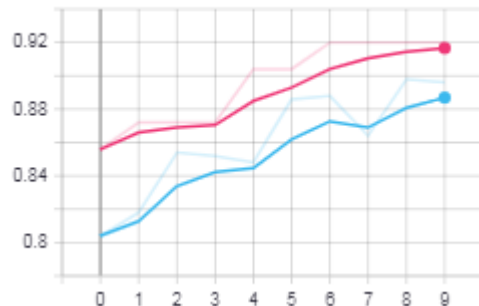
- **Conclusion :**
 - Surapprentissage

Name	Smoothed	Value	Step	Time	Relative
resnet50_2_run_2019_12_30-11_34_36/train	0.8972	0.88	9	Mon Dec 30, 13:00:18	22m 53s
resnet50_2_run_2019_12_30-11_34_36/validation	0.1828	0.1833	9	Mon Dec 30, 13:00:18	22m 53s

Xception

Batch size = 30

epoch_accuracy



epoch_loss



- Temps d'exécution : 1h 10min
- Précision : 98.17%

Conclusion :
Le modèle a moins de tendances à surapprendre et une meilleure précision même avec seulement 10 epochs

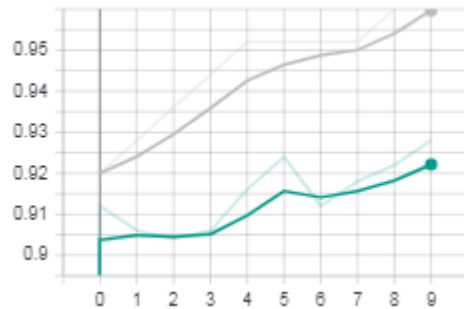
Name	Smoothed	Value	Step	Time	Relative
xception2_run_2019_12_30-14_27_37/train	0.8869	0.896	9	Mon Dec 30, 16:37:46	1h 2m 53s
xception2_run_2019_12_30-14_27_37/validation	0.9166	0.92	9	Mon Dec 30, 16:37:46	1h 2m 53s

Transfer Learning – Tests sur échantillon

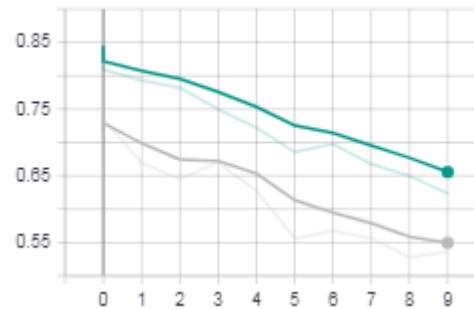
Xception – Fine tuning partiel

Nous gardons des poids des 1ères 7 couches (block 1)

epoch_accuracy



epoch_loss

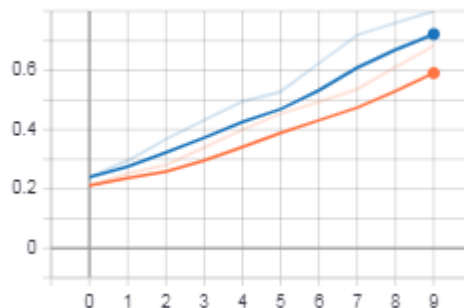


- Temps d'exécution : 1h 9min
- Précision : 98.78%

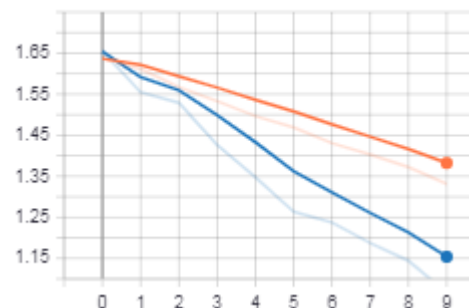
Name	Smoothed	Value	Step	Time	Relative
xception_FT1_run_2019_12_30-15_38_38/train	0.9222	0.928	9	Mon Dec 30, 18:00:55	1h 14m 57s
xception_FT1_run_2019_12_30-15_38_38/validation	0.9597	0.968	9	Mon Dec 30, 18:00:55	1h 14m 57s

Nous gardons des poids des 1ères 16 couches (block 1 + 2)

epoch_accuracy



epoch_loss



- Temps d'exécution : 1h 8min
- Précision : 81.71 %

Conclusion : Le meilleur modèle est Xception avec le fine-tuning en gardant le poids de 1^{er} block.

Name	Smoothed	Value	Step	Time	Relative
xception_FT2_run_2019_12_30-17_47_18/train	0.5911	0.682	9	Mon Dec 30, 19:55:43	1h 1m 19s
xception_FT2_run_2019_12_30-17_47_18/validation	0.7223	0.8	9	Mon Dec 30, 19:55:43	1h 1m 19s

5. COMPARAISON

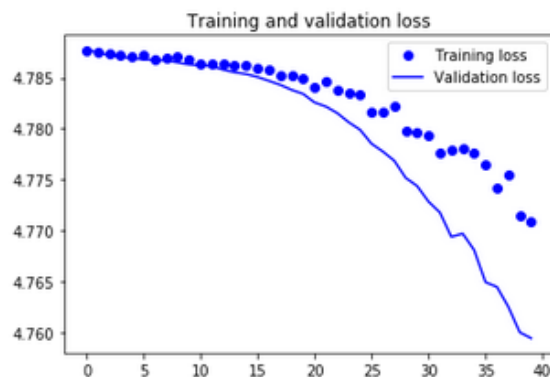
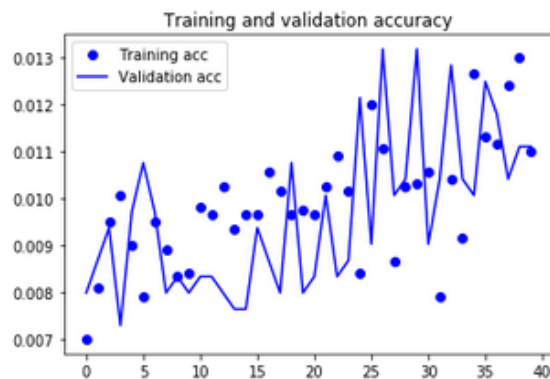
5. Comparaison de meilleurs modèles

CNN from scratch vs. Xception transfer learning

- Les modèles ont été entraînés sur les données complètes pendant 40 epochs dans l'environnement Sagemaker

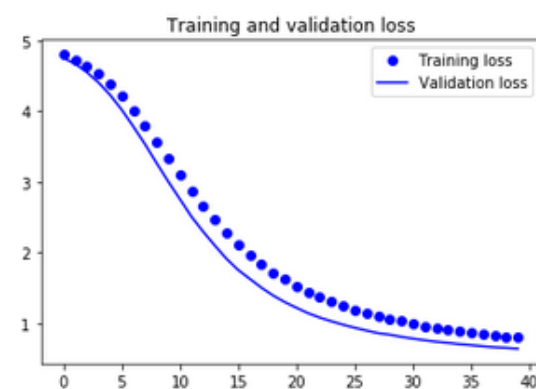
CNN from scratch

- Temps d'exécution : 1h 53min
- Précision : 0.79 %



Xception

- Temps d'exécution : 2h 0min
- Précision : 85.75 %



6. PRÉDICTIONS

6. Prédiction de race avec le modèle final

- Nos avons enregistré le modèle final Xception dans un fichier .h5
- Le modèle est loadé avec la commande suivante :

```
# Load the model
model = tf.keras.models.load_model('redim_download/xception2.h5')
```

- La fonction suivante prend en argument une photo qui est transformée en array, redimensionnée et passée dans le modèle pour la prédiction

```
def predict_breed(filename):
    """ Function which predicts the dog's breed and indicate the probability of the prediction.abs

    ARG : filename of the photo (ex. .jpg) or path to the photo in form of string
    OUTPUT : print the breed + its probability in %

    """

    # Load the image
    img = load_img(filename, target_size=(224, 224))
    # Convert to array
    img = img_to_array(img)
    # Reshape into a single sample with 3 channels
    img = img.reshape(1, 224, 224, 3)
    # Normalize the image and convert it to float
    img=img/255.
    # Predict the breed probabilities
    result = model.predict(img)
    # Indicate the position (number) of the most probable breed
    pos = np.argmax(result[0])
    # Indicate the path to the train data
    train_data_dir = pathlib.Path('redim_download/train')
    # Obtain class names
    CLASS_NAMES = np.array([item.name for item in train_data_dir.glob('*') if item.name != "LICENSE.txt"])
    # Return name of the most probable class
    print ("Ce chien est probablement un : " + str(CLASS_NAMES[pos][10:]))
    print ("La probabilité qu'il s'agit de cette race est : " + str(round(result[0][pos]*100,2)) + "%")
```

6. Prédiction de race avec le modèle final



Ce chien est probablement un : papillon
La probabilité qu'il s'agit de cette race est : 94.37%



Ce chien est probablement un : bluetick
La probabilité qu'il s'agit de cette race est : 99.27%



Ce chien est probablement un : chow
La probabilité qu'il s'agit de cette race est : 97.2%



Ce chien est probablement un : Irish_wolfhound
La probabilité qu'il s'agit de cette race est : 8.51%



7. CONCLUSION

7. Conclusion

- Le projet m'a permis de :
 - Me familiariser avec les CNN et leur architecture
 - Prendre en main le package Keras et Tensorflow
 - Comprendre l'intérêt d'organisation de travail et d'optimisation d'algorithmes lors qu'il s'agit de calcul exigeant en ressources
- Je n'ai pas réussi à obtenir un bon résultat en construisant un modèle à partir de scratch. Par contre, il est tout à fait possible d'obtenir des résultats intéressants pour l'application pratique en utilisant un modèle déjà existant et librement accessible
- Améliorations proposées :
 - Donner un meilleur cadre à optimisation de paramètres
 - Améliorer le modèle final avec le tuning de paramètres (ex. learning rate)

Liens & ressources

Voici les liens qui m'ont été utiles lors de travail sur ce projet :

<http://mccormickml.com/2014/07/24/intuition-behind-whitening-image-patches/>

<https://medium.com/datadriveninvestor/using-deep-learning-to-classify-breeds-of-dogs-from-images-2b026ea03436>

<https://medium.com/nanonets/how-to-easily-build-a-dog-breed-image-classification-model-2fd214419cde>

<https://towardsdatascience.com/dog-breed-classification-hands-on-approach-b5e4f88c333e>

<https://machinelearningmastery.com/image-augmentation-deep-learning-keras/>

<https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/>

https://www.tensorflow.org/tutorials/load_data/images

https://docs.aws.amazon.com/fr_fr/sagemaker/latest/dg/gs.html

<https://keras.io/preprocessing/image/>

<https://keras.io/models/sequential/>

<https://github.com/fchollet/deep-learning-with-python-notebooks>

<https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/>

<https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d#676b>

Ainsi qu'un vrai livre :

Aurélien Géron : Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow