



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

Morgan May

Projet n°6 du parcours Ingénieur Machine Learning





Sommaire

1. Sujet & interprétation
2. Exploration et prétraitement des données
3. Modélisation et itérations de développement
4. Modélisation par une approche de transfer learning
5. Conclusion et pistes d'amélioration



1. Sujet & interprétation



1. Sujet & interprétation

Le sujet de ce projet est la réalisation d'un algorithme de détection de la race d'un chien sur une photo.

Pour réaliser ce projet, nous utiliserons le dataset suivant contenant 20580 images de chiens réparties en 120 races : <http://vision.stanford.edu/aditya86/ImageNetDogs/>

Ce sujet nous permet de prendre en main un exemple concret de projet de vision par ordinateur et des techniques de Deep Learning associées.



2. Exploration et prétraitement des données



2. Exploration et prétraitement des données

L'exploration des images de jeu de données, nous permet de réaliser plusieurs observations que l'on va exploiter pour notre préparation de données :

- Les images sont très diverses en termes de cadrage, de sujet... :
 - il est possible de retrouver plusieurs chiens sur une même photo, certains chiens d'une même race n'ont pas les mêmes couleurs ou sont adultes ou chiots...
 - il est possible trouver un humain ou d'autres objets en plus d'un chien, certains chiens portent des "habits"
 - certaines photos sont prises de loin tandis que d'autres sont des portraits...
- Les images sont de tailles très variées.
- Elles ont aussi, naturellement, des expositions et réglages de contraste différents.



2. Exploration et prétraitement des données

Globalement, ces observations nous permettent de décrire un **problème assez complexe à résoudre**, d'autant plus que le **nombre de classes parmi lesquelles prédire est conséquent (120)** et que le **dataset est relativement restreint (un peu plus de 20000 images)**.

En choisissant de rendre ces tâches de prétraitement complètement automatisées, nous allons ici en effectuer principalement 3 :

- La manipulation d'histogramme pour régler l'exposition et le contraste
- Le réglage des tailles des images
- L'augmentation artificielle de données

Les deux premières étapes, détaillées dans les slides suivantes, sont réalisées sur l'ensemble des données puis celles-ci sont partagées en trois datasets (train set : 80%, validation set : 10%, test set : 10%) en faisant attention à bien conserver **les mêmes proportions pour chacune des races**.

2. Exploration et prétraitement des données

Manipulation d'histogramme :

La manipulation de l'histogramme des intensités des pixels nous permet de régler l'exposition et le contraste des images. Du point de vue de notre algorithme, cela effectue une forme de normalisation qui est intéressante pour l'efficacité du processus d'apprentissage.

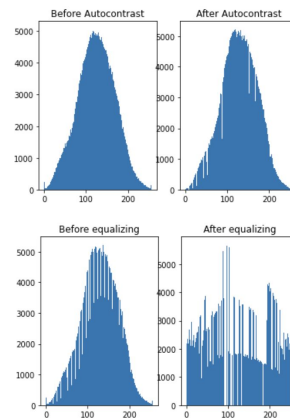
Pour réaliser ces opérations, on a ici utilisé le module *Pillow* et les fonctions *autocontrast()* et *equalize()* de son API *ImageOps*.

Note : ici, on a finalement montré que ces étapes n'avait globalement pas d'impact sur les performances obtenues sur la baseline.

Réglage des tailles des images :

Les modèles de deep learning (réseaux de neurones) que nous allons mettre en œuvre vont utiliser une architecture fixée une fois le modèle établi. Concrètement, la couche d'entrée du réseau de neurones contiendra toujours le même nombre de "neurones". Il faut donc que le nombre d'entrées soit constant, c'est-à-dire que la taille des images soit identique pour chaque image.

Ici, on a testé plusieurs tailles d'images : 128x128, 256x256 puis 224x224



Exemple de modification de l'histogramme d'intensité d'une image

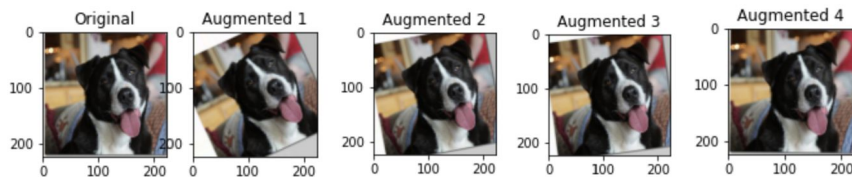
2. Exploration et prétraitement des données

Data augmentation :

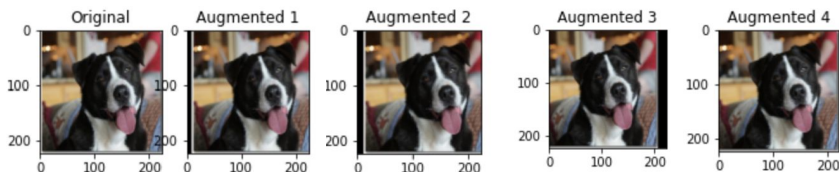
L'augmentation artificielle de données est une technique consistant à créer de nouvelles images artificiellement à partir des images dont l'on dispose déjà afin d'augmenter la taille du dataset. Nous avons ici utilisé les fonctions déjà implémentées dans la fonction `ImageDataGenerator()` de l'API preprocessing de keras. Concrètement, cette fonction permet de présenter à notre modèle une version légèrement modifiée de chaque image pour chaque époque.

Voici les principales modifications apportées de manière aléatoire aux images du dataset afin de créer des images superficielles supplémentaires sur lesquelles entraîner aussi nos modèles :

- Rotation :



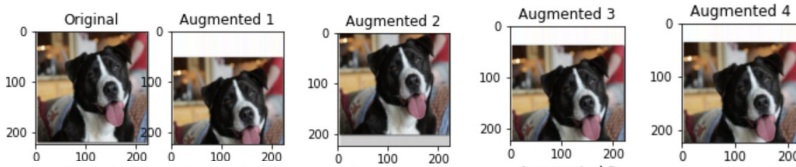
- Translation horizontale :



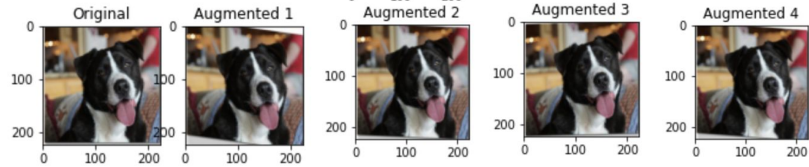
2. Exploration et prétraitement des données

suite Data augmentation :

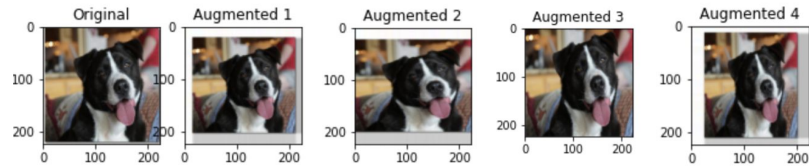
- Translation verticale :



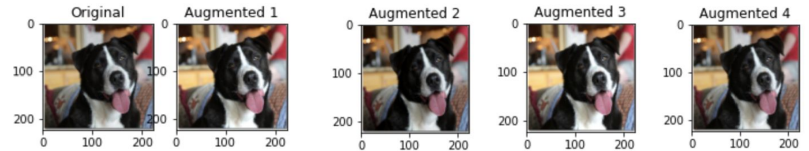
- Cisaillement :



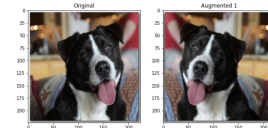
- Zoom :



- Intensité / Teinte :



- Mirroring :





3. Modélisation et itérations de développement



3. Modélisation et itérations de développement

Pour réaliser ce projet de classification d'images, on a choisi de développer un **modèle de type réseau de neurones à convolutions**. Ce type de réseau est globalement constitué de 2 parties : les premières couches (les plus profondes) réalisent des opérations de convolutions dont l'objectif est l'extraction de "features" tandis que la seconde partie est généralement constituée de couches denses dont l'objectif est la prise de décision pour la classification.

Nous avons ainsi réalisé une baseline avec Keras et TensorFlow, dotée de 16,8M de paramètres :

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 128, 128, 32)	896
max_pooling2d_1 (MaxPooling 2D)	(None, 64, 64, 32)	0
flatten_1 (Flatten)	(None, 131072)	0
dense_2 (Dense)	(None, 128)	16777344
dense_3 (Dense)	(None, 120)	15480

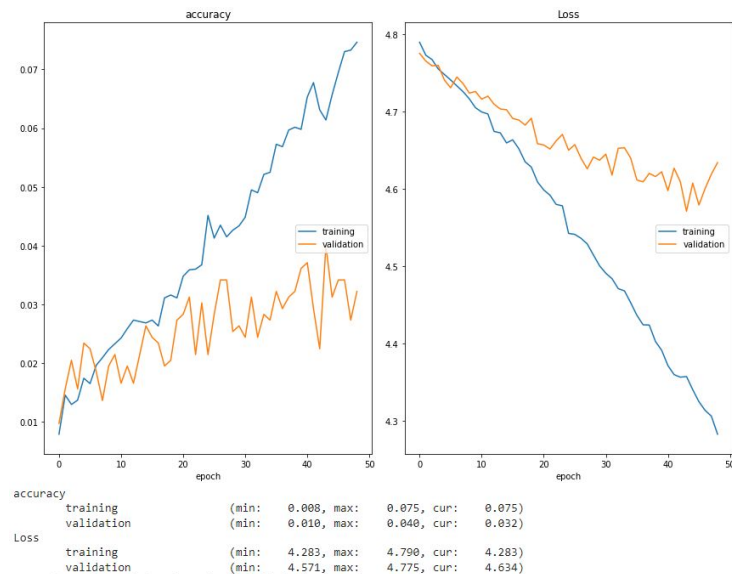
=====
Total params: 16,793,720
Trainable params: 16,793,720
Non-trainable params: 0

Celle-ci est composée :

- d'une couche de convolution de 32 filtres de 3x3 avec une activation RELU et un padding 'SAME'
- d'un Max pooling de 2x2 avec un pas horizontal et vertical de 2
- d'une couche de vectorisation ("flatten")
- d'une couche dense composée de 128 unités ("neurones") avec une fonction d'activation RELU également
- d'une couche softmax dotée de 120 unités correspondantes au nombre de classes à prédire

3. Modélisation et itérations de développement

Pour l'entraînement de notre modèle, nous choisissons dans un premier temps un optimiseur de type Gradient Descendant Stochastique (SGD) et "l'accuracy" comme métrique pour ce problème de classification. Nous utilisons initialement un batch-size de 32, un learning rate de 0.001 et un early stopping avec une patience de 5. Les performances suivantes sont observées sur la baseline :



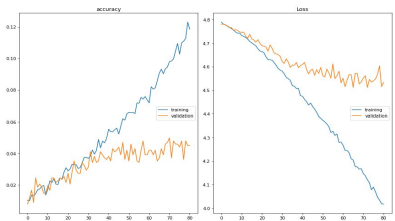
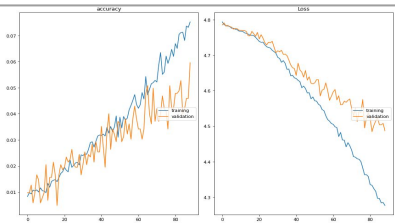
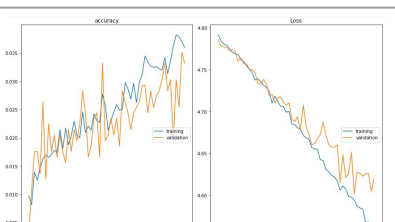
Les premiers constats que l'on fait sont les suivants :

- Notre modèle est globalement peu performant (4% sur le validation set)
- Il y a un surajustement important du training set
- L'entraînement est relativement peu efficace

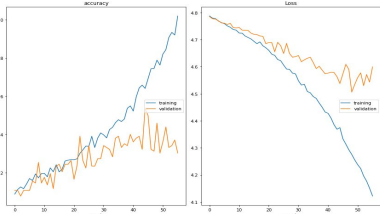
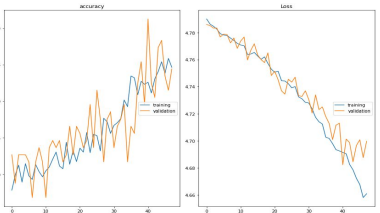
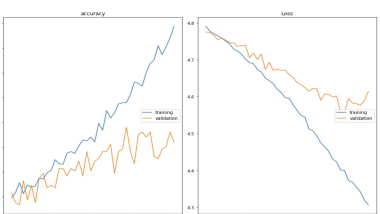
La première itération de développement que nous ferons sera donc d'essayer de régulariser ce modèle par l'intermédiaire d'un "dropout".

Les slides suivantes montrent les étapes de développement du modèle qui se sont surtout focalisées sur l'essai d'ajustement de l'ensemble des hyperparamètres.

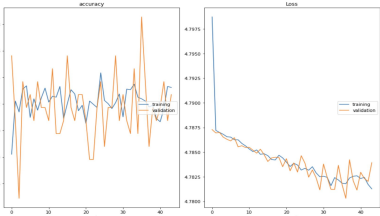
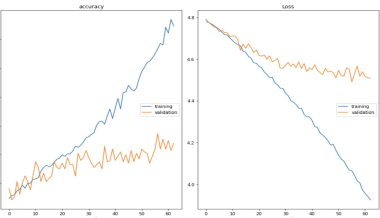
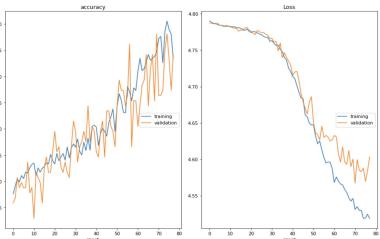
3. Modélisation et itérations de développement

Itération	Ajustements	Courbes d'apprentissage	Résultats sur les métriques	Observations
1	Ajout d'un Dropout léger (20%) sur la couche dense de 128 neurones précédant la couche softmax.		<p>Nombre de Paramètres : 16,8M</p> <p>accuracy training (min: 0.010, max: 0.123, cur: 0.119) validation (min: 0.008, max: 0.050, cur: 0.045)</p> <p>Loss training (min: 4.017, max: 4.789, cur: 4.017) validation (min: 4.511, max: 4.781, cur: 4.533)</p>	Il y a toujours un fort surajustement avec un dropout à 20% mais il a permis au modèle d'améliorer légèrement ses performances (4.5% sur le validation set contre 4%)
2	Test du changement de la taille du MaxPooling : passé de 2x2 à 6x6.		<p>Nombre de Paramètres : 15,7M</p> <p>accuracy training (min: 0.008, max: 0.075, cur: 0.075) validation (min: 0.005, max: 0.060, cur: 0.060)</p> <p>Loss training (min: 4.277, max: 4.794, cur: 4.277) validation (min: 4.485, max: 4.791, cur: 4.487)</p>	Ce changement (combiné avec le Dropout à 20%) a permis de régulariser le modèle (moins d'inputs pour les couches décisionnelles) et donc d'atteindre une accuracy de 8% sur le validation set
3	Test du changement du pas du MaxPooling : passé de (2, 2) à (4, 4). (combiné aux 2 itérations précédentes)		<p>Nombre de Paramètres : 3,95M</p> <p>accuracy training (min: 0.008, max: 0.038, cur: 0.036) validation (min: 0.003, max: 0.035, cur: 0.033)</p> <p>Loss training (min: 4.553, max: 4.792, cur: 4.558) validation (min: 4.602, max: 4.786, cur: 4.620)</p>	Augmenter le pas du MaxPooling semble ici avoir fait perdre de l'information utile aux couches décisionnelles.

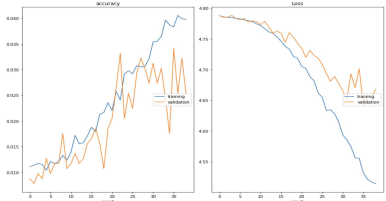
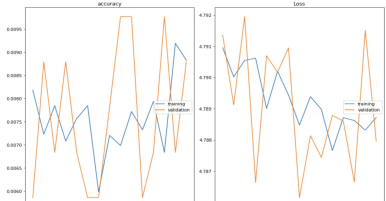
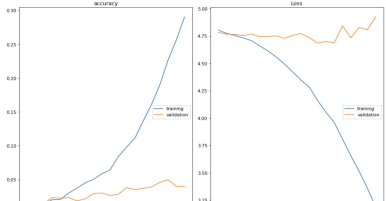
3. Modélisation et itérations de développement

Itération	Ajustements	Courbes d'apprentissage	Résultats sur les métriques	Observations
4	Test de modification de la taille des filtres : de 3x3 à 9x9 (avec dropout de 20% mais en revenant à la baseline pour le MaxPooling).		<p>Nombre de Paramètres : 16,8M</p> <pre> accuracy training (min: 0.000, max: 0.102, cur: 0.102) validation (min: 0.000, max: 0.054, cur: 0.030) Loss training (min: 4.122, max: 4.788, cur: 4.122) validation (min: 4.506, max: 4.785, cur: 4.600) </pre>	Comme pour le changement de la taille du MaxPooling, il y a un léger effet de régularisation
5	Test de modification du nombre de filtres : de 32 à 128 (avec Dropout à 33%)		<p>Nombre de Paramètres : 67,1M</p> <pre> accuracy training (min: 0.008, max: 0.026, cur: 0.025) validation (min: 0.007, max: 0.031, cur: 0.024) Loss training (min: 4.658, max: 4.790, cur: 4.661) validation (min: 4.682, max: 4.786, cur: 4.700) </pre>	Les performances atteintes sont moins bonnes, même par rapport à la baseline, comme si ce modèle générait trop d'informations sans que ses couches décisionnelles ne sachent quoi en faire
6	Test de modification des fonctions d'activation : utilisation de tangente hyperbolique (tanh) au lieu de RELU		<p>Nombre de Paramètres : 16,8M</p> <pre> accuracy training (min: 0.010, max: 0.079, cur: 0.079) validation (min: 0.007, max: 0.038, cur: 0.032) Loss training (min: 4.307, max: 4.791, cur: 4.307) validation (min: 4.557, max: 4.776, cur: 4.614) </pre>	Le changement de fonction d'activation n'a pas affecté particulièrement les résultats par rapport à la baseline

3. Modélisation et itérations de développement

Itération	Ajustements	Courbes d'apprentissage	Résultats sur les métriques	Observations
7	Test de modification de la fonction d'optimisation : utilisation d'Adam au lieu d'un SGD		<p>Nombre de paramètres : 16,8M</p> <pre> accuracy training (min: 0.008, max: 0.014, cur: 0.013) validation (min: 0.005, max: 0.019, cur: 0.013) Loss training (min: 4.781, max: 4.799, cur: 4.781) validation (min: 4.780, max: 4.787, cur: 4.784) </pre>	L'algorithme a eu du mal à apprendre même après réglage du learning rate
8	Augmentation de la "largeur" de la couche dense : 256 unités au lieu de 128 (en conservant 20% de Dropout)		<p>Nombre de paramètres : 33,5M</p> <pre> accuracy training (min: 0.010, max: 0.134, cur: 0.129) validation (min: 0.009, max: 0.055, cur: 0.048) Loss training (min: 3.926, max: 4.790, cur: 3.926) validation (min: 4.491, max: 4.778, cur: 4.509) </pre>	Logiquement, le même soucis de surajustement du jeu d'entraînement est visible mais les performances sur le jeu de validation sont tout de même légèrement meilleures que la baseline (5,5%)
9	Augmentation de la "profondeur" des couches de convolution : 4 couches identiques de convolution + MaxPooling		<p>Nombre de paramètres : 584k</p> <pre> accuracy training (min: 0.008, max: 0.041, cur: 0.033) validation (min: 0.003, max: 0.038, cur: 0.034) Loss training (min: 4.519, max: 4.790, cur: 4.519) validation (min: 4.567, max: 4.787, cur: 4.603) </pre>	Les performances sont similaires à la baseline mais le surajustement n'est plus présent : c'est globalement un modèle plus efficient

3. Modélisation et itérations de développement

Itération	Ajustements	Courbes d'apprentissage	Résultats sur les métriques	Observations
10	Adaptation des nouvelles couches de convolution : couche de 256 filtres, couche de 128, MaxPooling, couche de 32, MaxPooling		<p>Nombre de paramètres : 4,55M</p> <p>accuracy training (min: 0.010, max: 0.041, cur: 0.040) validation (min: 0.008, max: 0.034, cur: 0.025)</p> <p>Loss training (min: 4.514, max: 4.788, cur: 4.514) validation (min: 4.642, max: 4.788, cur: 4.668)</p>	L'adaptation des couches de convolution ajoutée n'a quasiment pas eu d'impact sur l'apprentissage et les performances atteintes.
11	Test du remplacement des couches denses par un GlobalAveragePooling (architecture baseline sinon : 1 couche de convolution...)		<p>Nombre de paramètres : 4856</p> <p>accuracy training (min: 0.006, max: 0.009, cur: 0.009) validation (min: 0.006, max: 0.010, cur: 0.009)</p> <p>Loss training (min: 4.788, max: 4.791, cur: 4.789) validation (min: 4.786, max: 4.792, cur: 4.788)</p>	Logiquement, le remplacement de la couche dense (et donc de la plupart des paramètres) a rendu ce modèle clairement pas assez complexe et sous-ajuste largement le jeu de données d'entraînement.
12	Test d'ajout d'une normalisation en sortie de la couche de convolution et avant l'activation		<p>Nombre de paramètres : 16,8M</p> <p>accuracy training (min: 0.010, max: 0.290, cur: 0.290) validation (min: 0.010, max: 0.050, cur: 0.040)</p> <p>Loss training (min: 3.188, max: 4.803, cur: 3.188) validation (min: 4.684, max: 4.924, cur: 4.924)</p>	La normalisation semble avoir un impact fort sur les capacités du modèle à apprendre mais ça n'a ici que peu d'impact sur les performances du fait d'un très grand sur-ajustement

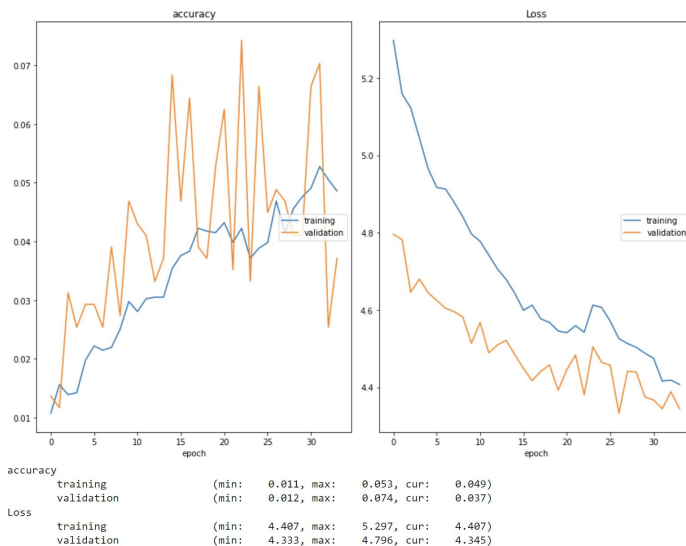
3. Modélisation et itérations de développement

En réutilisant les résultats les plus probants des expérimentations précédentes pour construire un modèle plus performant et en ajustant le learning rate et le batch-size, nous obtenons le modèle et les résultats suivants :

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 32)	7808
batch_normalization (Batch Normalization)	(None, 128, 128, 32)	128
activation (Activation)	(None, 128, 128, 32)	0
max_pooling2d (MaxPooling2D)	(None, 62, 62, 32)	0
dropout (Dropout)	(None, 62, 62, 32)	0
flatten (Flatten)	(None, 123008)	0
dense (Dense)	(None, 256)	31490304
batch_normalization_1 (Batch Normalization)	(None, 256)	1024
activation_1 (Activation)	(None, 256)	0
dropout_1 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 256)	65792
batch_normalization_2 (Batch Normalization)	(None, 256)	1024
activation_2 (Activation)	(None, 256)	0
dropout_2 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 120)	30840

Ce modèle combine ainsi les modifications les plus prometteuses : normalisation, 2 couches denses, dropout, largeur de filtre plus importante (9x9).



Cette combinaison d'ajustements nous a ainsi permis d'améliorer les performances par rapport à la baseline (accuracy à **7.4% sur le validation set**).



4. Modélisation par transfer learning



4. Modélisation par transfer learning

Principe

Dans le but d'améliorer nettement les performances, nous allons probablement devoir changer d'architecture de modèle. D'autre part, notre dataset étant relativement petit (20k images); nous allons utiliser un modèle déjà pré-entraîné sur un dataset bien plus grand.

Ici, nous choisissons d'utiliser un modèle de l'état de l'art, un ResNet-50, pré-entraîné sur ImageNet (un dataset de plus d'un million d'images variées).

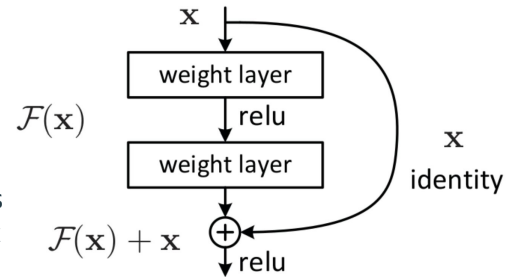
L'idée du procédé de transfer learning que nous mettons en œuvre ici est de ré-exploiter l'ensemble de la partie d'extraction de features (les couches de convolution, avec leurs paramètres entraînés) et de rajouter par-dessus une ou plusieurs couches "décisionnelles". Ainsi, on peut espérer que ce modèle pré-entraîné est capable de détecter des "features" de bas niveau (coins, bords voire pattes, oreilles...) pertinentes pour notre problème de classification de races de chien.

4. Modélisation par transfer learning

ResNet-50

Le modèle ResNet-50 mis en oeuvre ici est un modèle de deep learning disposant d'une architecture particulièrement profonde : il est composé de 50 couches qui sont des séquences de couches de convolution avec batch-normalisation, activation ReLU et MaxPooling.

Sa particularité réside dans son utilisation de connections résiduelles entre plusieurs blocs de convolution afin de parer au problème de disparition du gradient dans les réseaux profonds lors de l'entraînement.

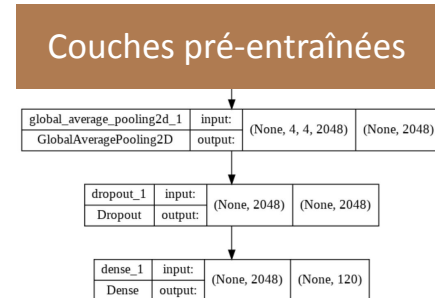


Source : iq.opengenus.org

Modifications / couches décisionnelles :

Au-dessus des couches pré-entraînées (et dont nous figeons les valeurs des paramètres pour l'entraînement sur notre dataset), nous ajoutons simplement une couche de pooling (de type GlobalAveragePooling), un Dropout à 20% et la couche dense softmax avec ses 120 unités.

A noter également que les étapes de préparation des données sont ici différentes car on réutilise les mêmes qui ont été mises en oeuvre pour pré-entraîner ce modèle à travers la fonction `preprocess_input()` du répertoire dédié de Keras. Celle-ci convertit les images RGB en BGR et réalise un centrage à 0 pour chaque couleur.

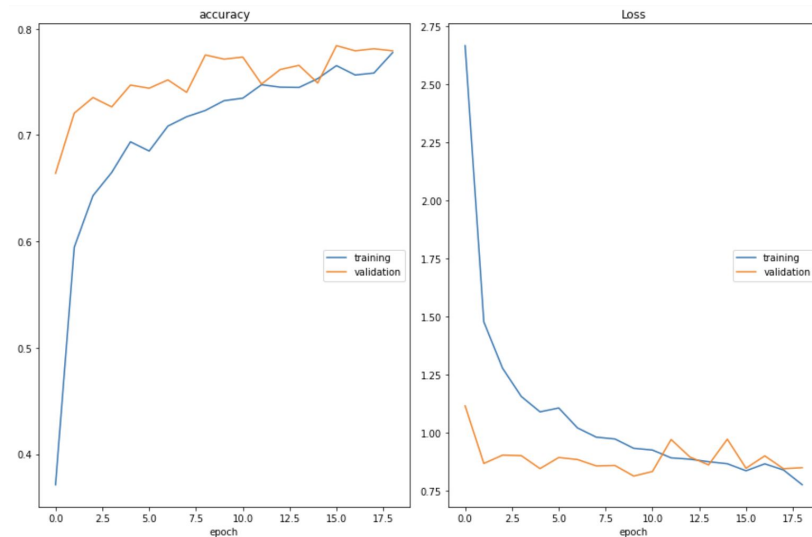


4. Modélisation par transfer learning

Entraînement et résultats

Comme on peut le voir ci-contre, **l'entraînement des couches supérieures est rapide et efficace** et les résultats obtenus avec ce modèle pré-entraîné (**78.4% sur le validation set et 72.2% sur le test set**) sont incomparables avec les résultats obtenus avec les modèles développés plus tôt.

Nous avons également pu montrer que **l'influence du pré-entraînement est très importante ici** car l'entraînement de ce même modèle directement sur notre dataset s'est révélé peu efficace et a atteint des performances relativement similaires à celles obtenues avec nos premiers modèles.



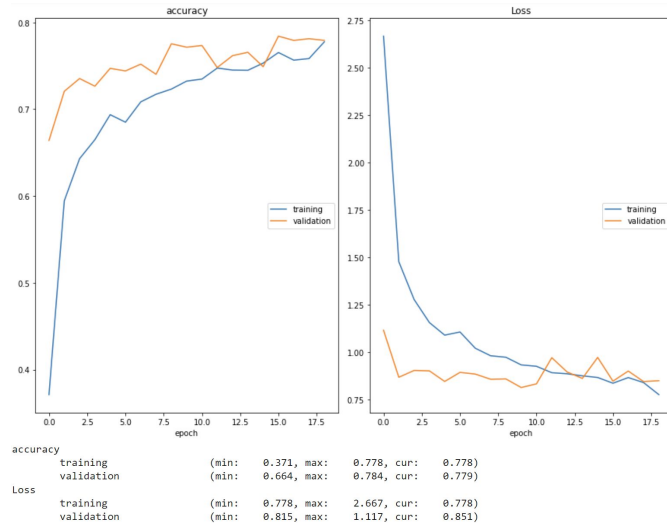
accuracy	training	(min: 0.371, max: 0.778, cur: 0.778)
	validation	(min: 0.664, max: 0.784, cur: 0.779)
Loss	training	(min: 0.778, max: 2.667, cur: 0.778)
	validation	(min: 0.815, max: 1.117, cur: 0.851)

4. Modélisation par transfer learning

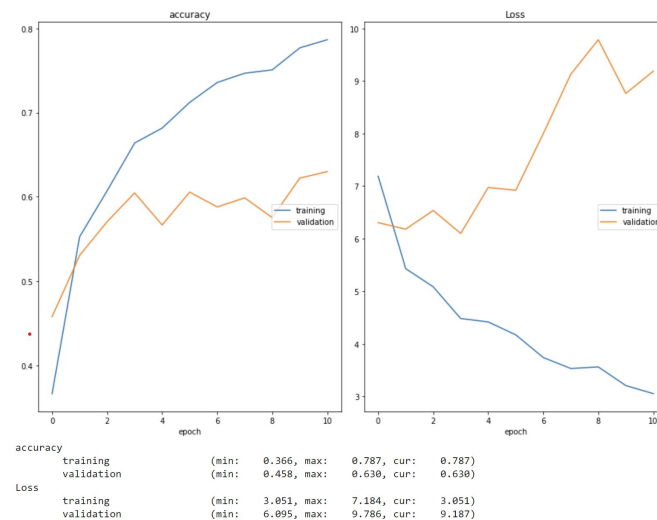
Influence de la Data augmentation

Comme on peut le voir ci-dessous, l'utilisation de la technique d'augmentation de données a eu également un impact important sur les performances obtenues en ayant eu un **effet de régularisation** :

Avec Data Augmentation :



Sans Data Augmentation :





5. Conclusion et pistes d'amélioration



5. Conclusion et pistes d'amélioration

Nous avons ainsi créé un modèle (mis à disposition via l'application fournie dans le même répertoire de fichiers que cette présentation) capable de prédire la race d'un chien apparent sur une photo parmi 120 races reconnues avec une probabilité de réussite d'environ 72%, ce qui paraît relativement suffisant pour aider à une prise de décision mais pas suffisant pour automatiser ce processus.

Bien que l'objectif de ce projet n'était pas d'optimiser au maximum les performances obtenues, voici quelques pistes d'amélioration envisageables :

- Retirer des données d'entraînement les images qui sont peu pertinentes, par exemple : si le chien porte un habit, si le chien n'est qu'un parmi plusieurs sujets de la photo...
- Essayer d'autres architectures et modèles pré-entraînés (notamment RandomForest, Xception...)
- Essayer un ré-entraînement léger (avec un learning rate très faible) des couches de convolution du modèle pré-entraîné
- Optimiser un peu plus l'entraînement des couches décisionnelles ajoutées au modèle pré-entraîné



Merci pour votre attention

