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

Open Classroom

Benjamin Bouchard

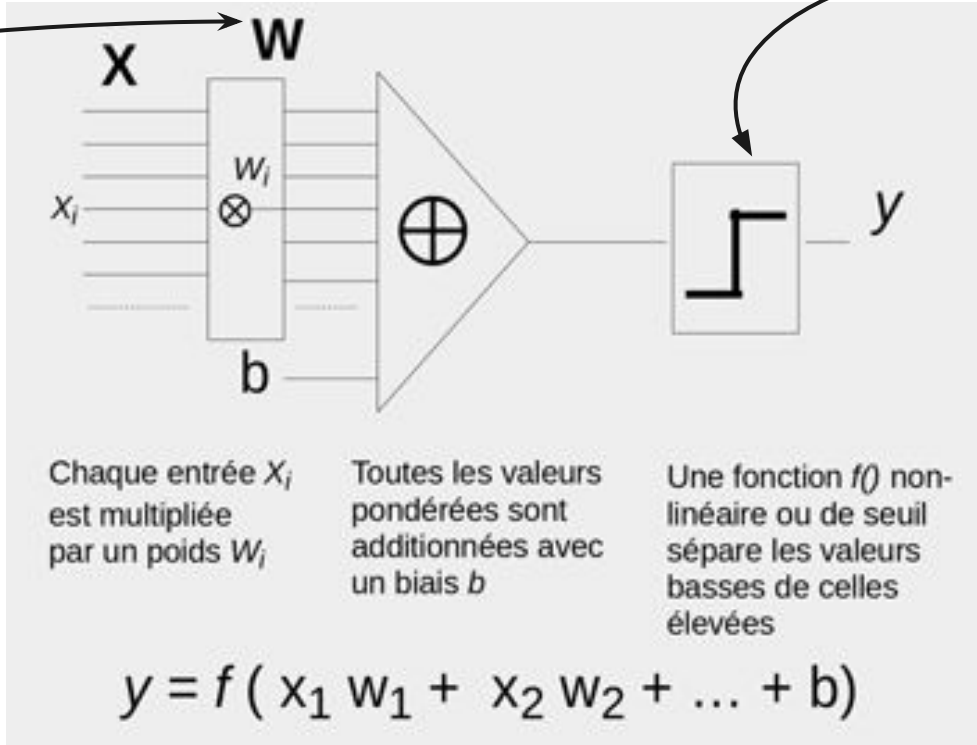
Introduction

- Déterminer la race d'un chien à partir d'une photo.
- Standford dogs dataset (issue de ImageNet)
    - 20 850 images
    - 120 races (breed)
- Problème de classification supervisée (données avec étiquettes)
- Données entrée: image ⇒ Vision par ordinateur (Computer Vision)

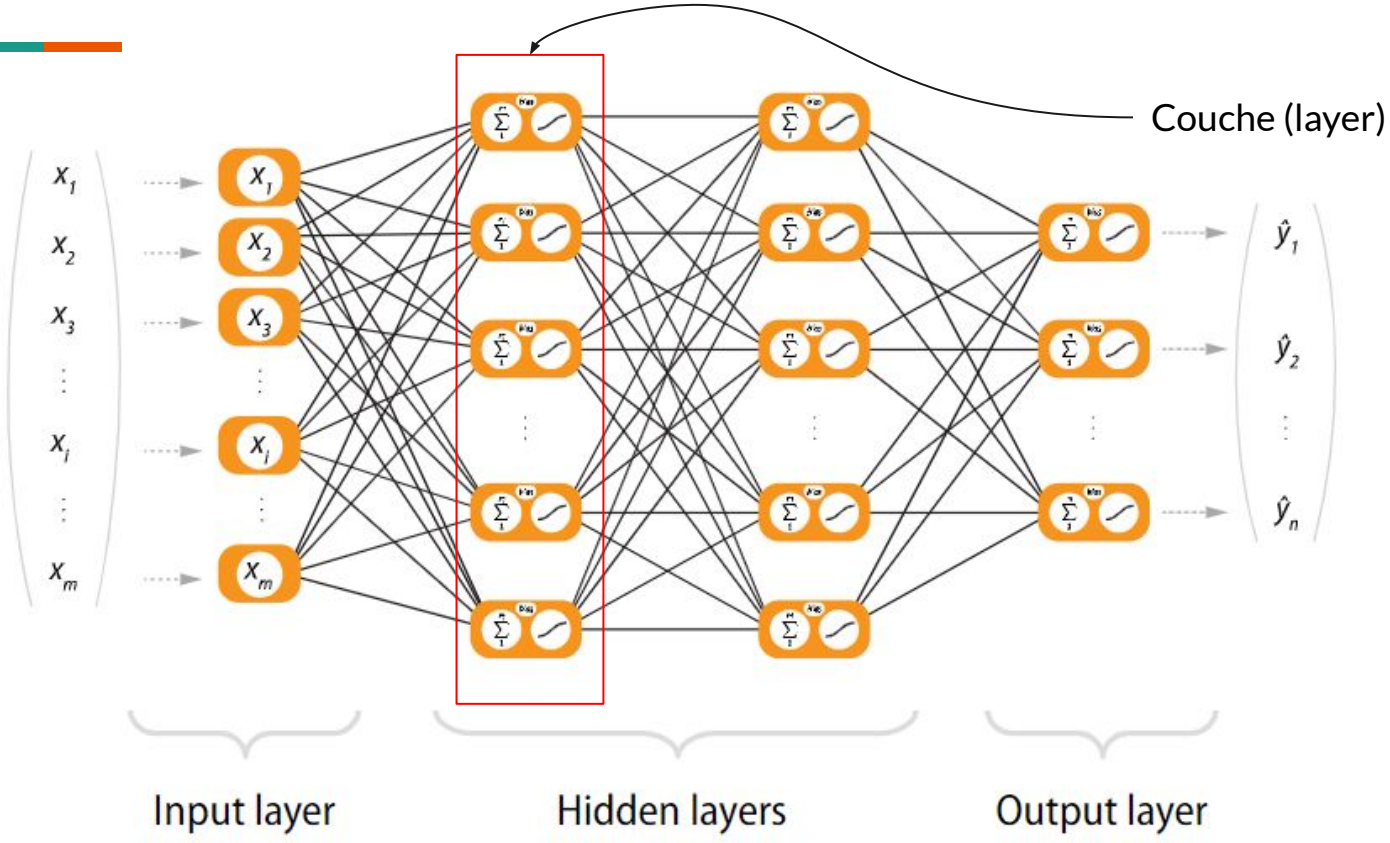
Réseau de neurones: élément de base

La fonction f est appelée **la fonction d'activation**

L'ensemble des poids (wi) est le **kernel** du neurone

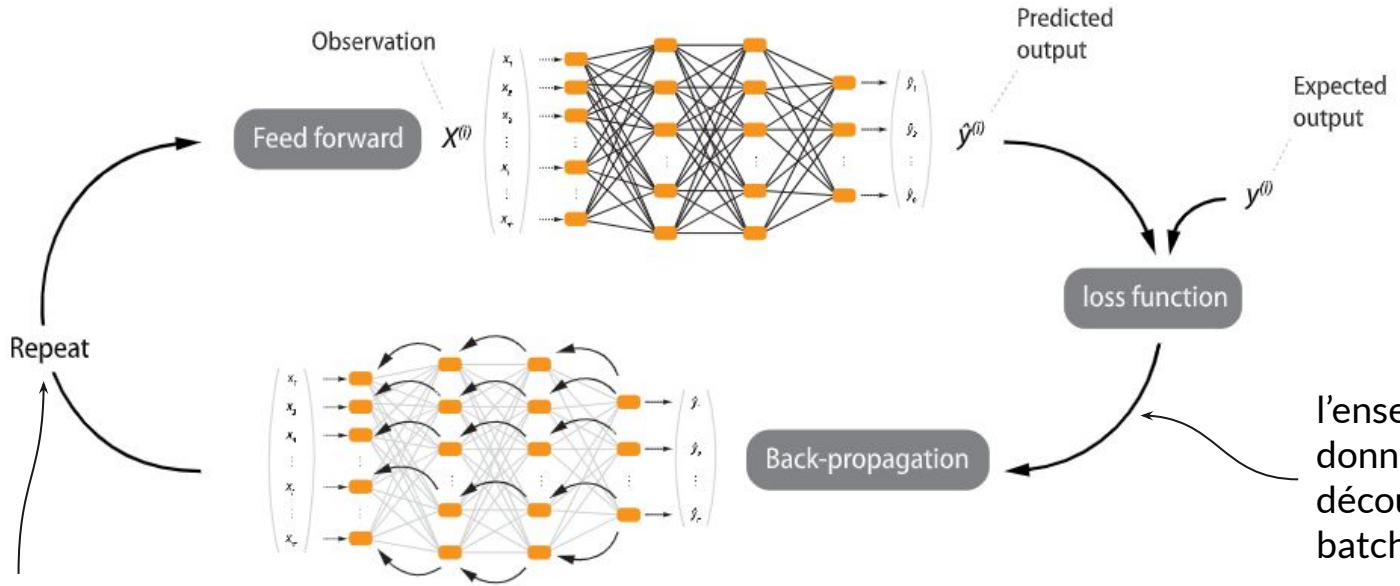


Réseau de neurones: empilement de couches



Visuel extrait de la formation FIDLE du CNRS: CNRSFormationFIDLE

Réseau de neurones: apprentissage

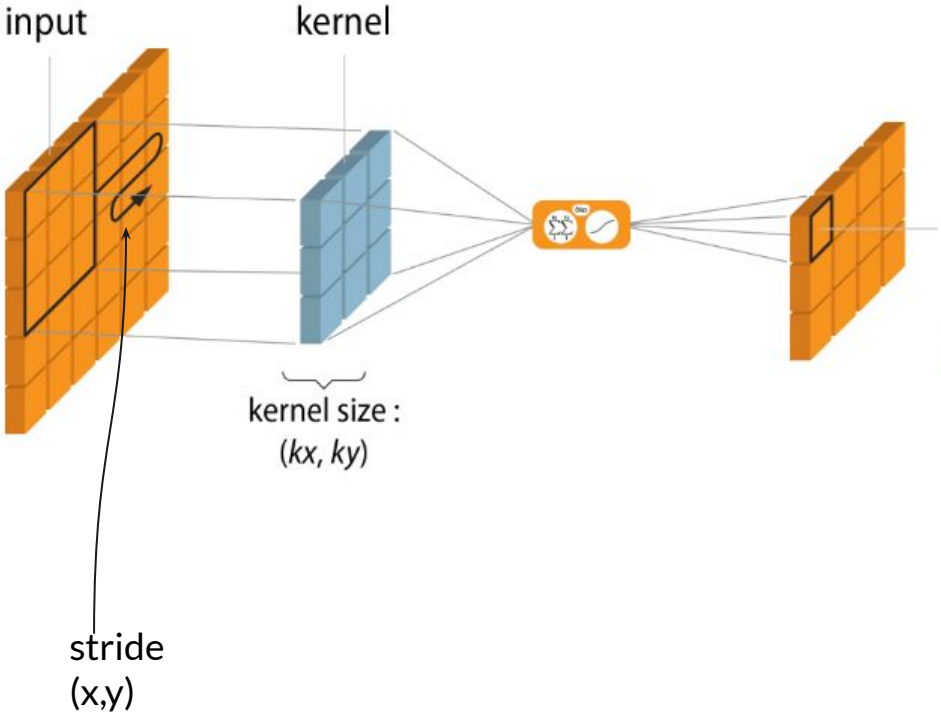


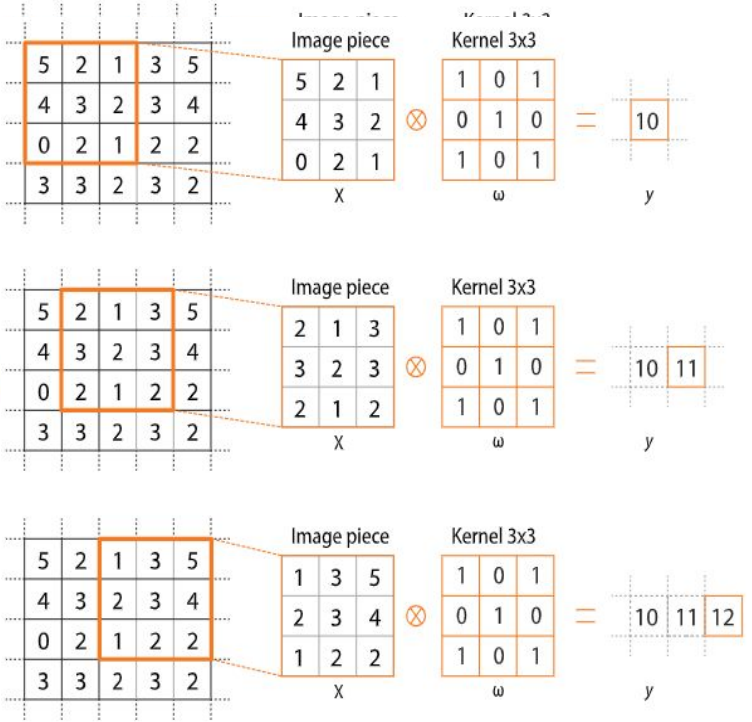
l'apprentissage est répété plusieurs fois (epochs)

Optimizer et learning rate (r)

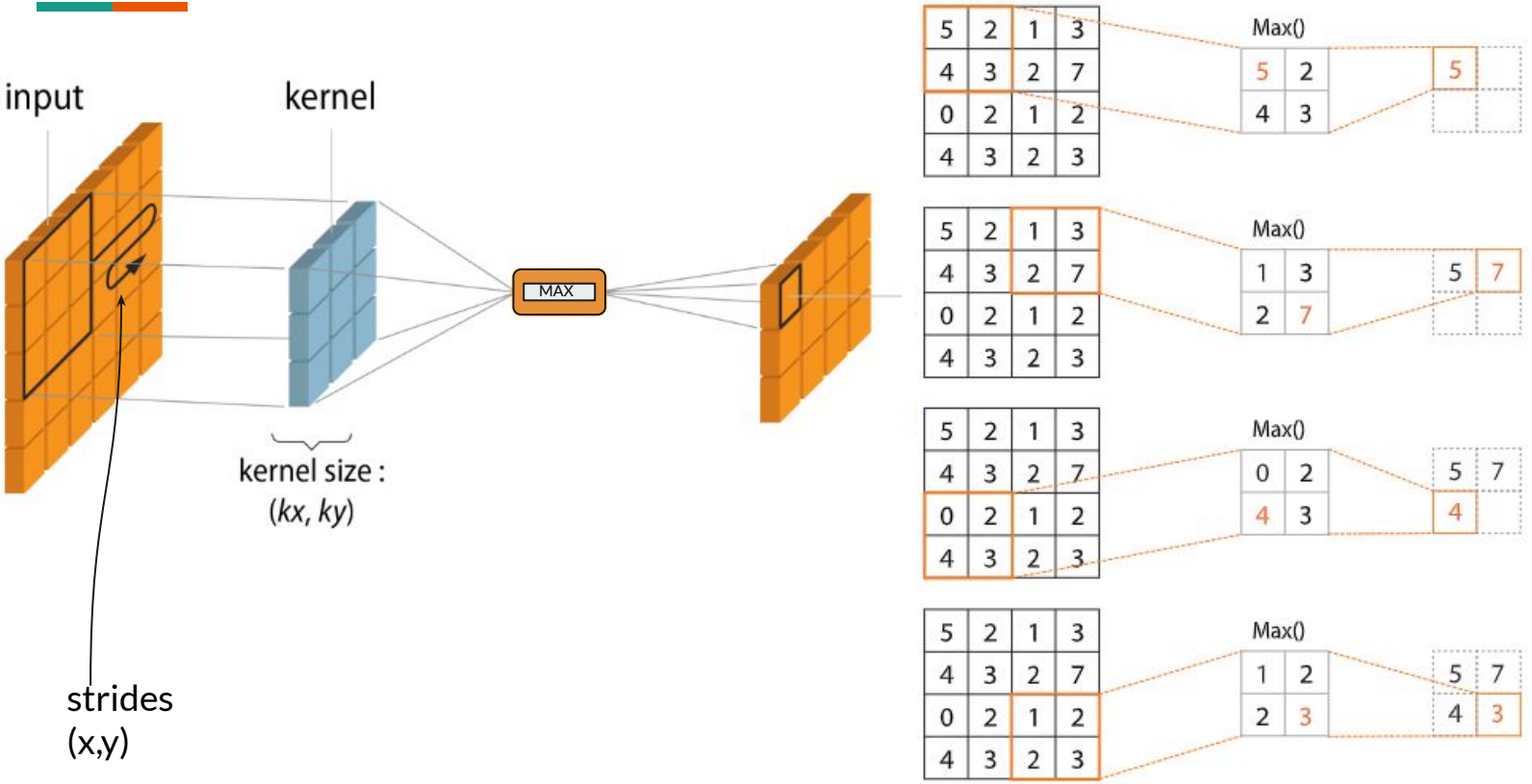
l'ensemble des données d'entrée est découpé en lot (taille batch size)

Réseau de convolution: convolution

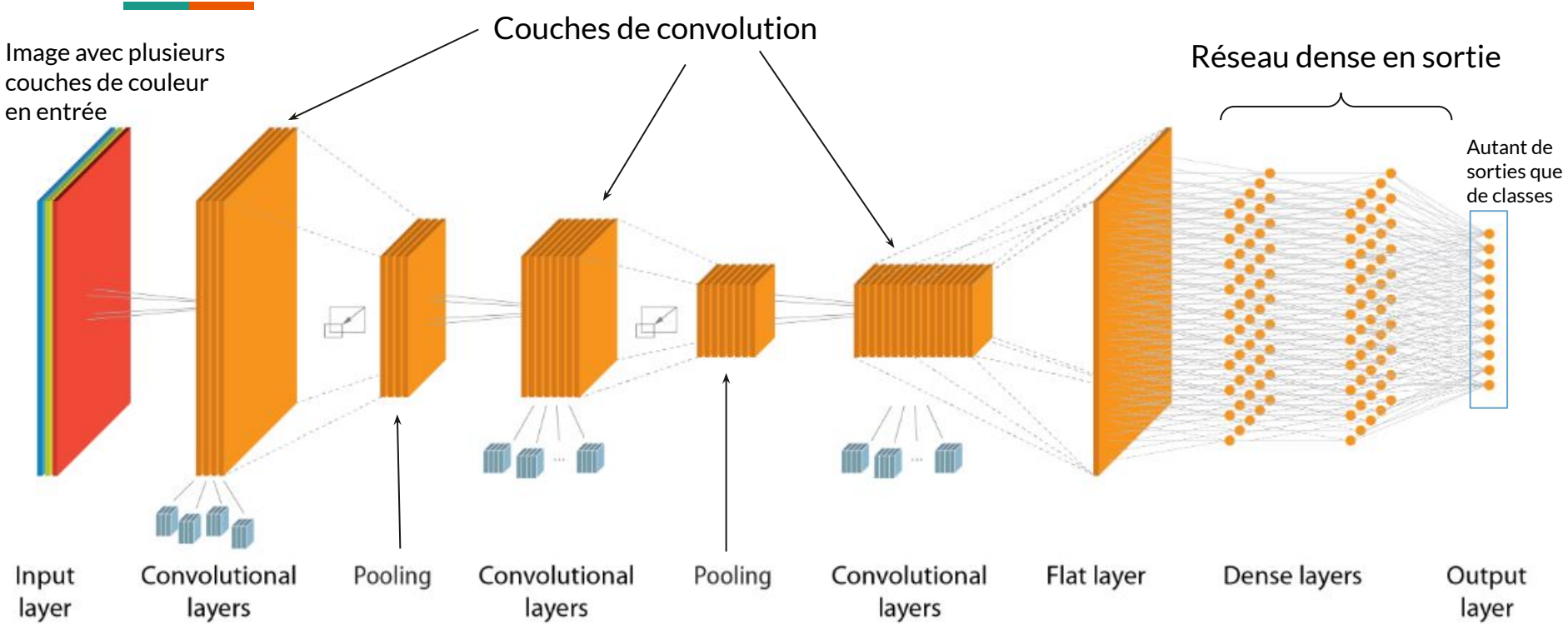




Réseau de convolution: max pooling

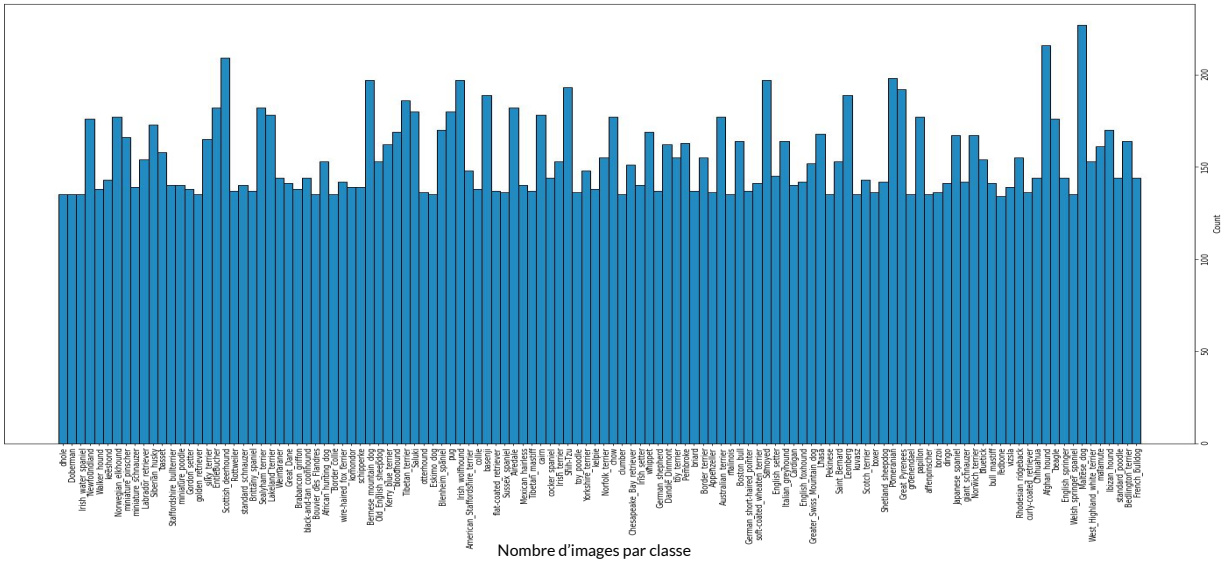


Réseau de convolution: traitement d'images et classification



Analyse des images

- Répartition des images par classe

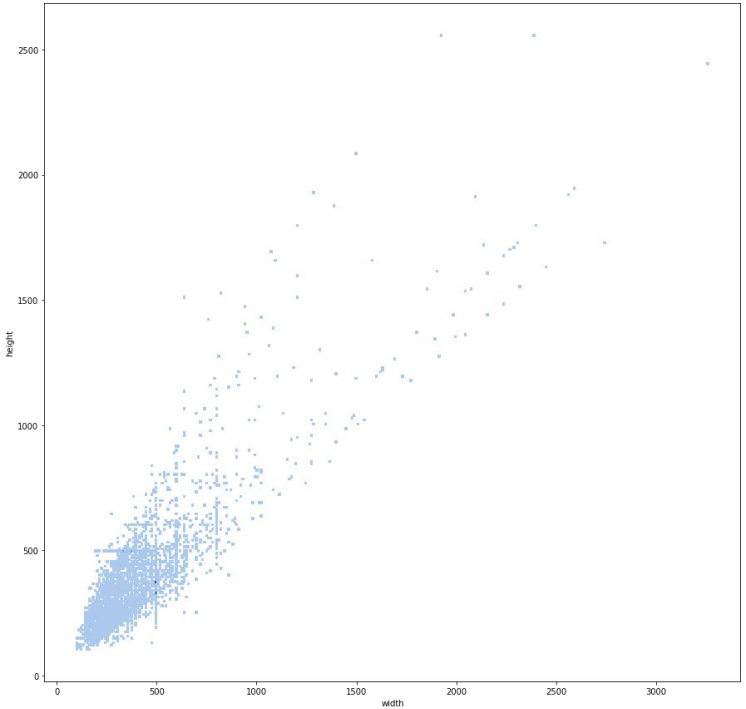


Nombre d'images par classe

- peu d'images par race: moyenne de 154 photos

Analyse des images

- Analyse des tailles



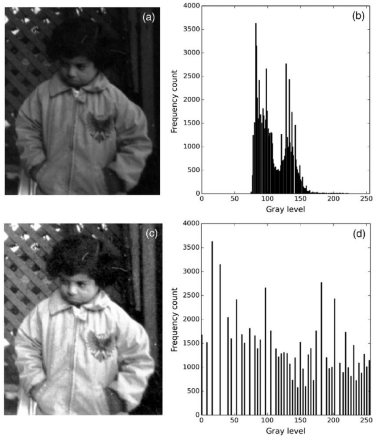
- Analyse des couches de couleurs

```
mode
RGB      20579
RGBA         1
dtype: int64
```

- Une image a écartée

Preprocessing des images

- Trois pré-traitements mis en place:
  - Resizing en 224x224x3
    - nécessaire pour les modèles VGG16
    - directement pris en charge par la routine TF de lecture des
  - Equalisation sur chaque couche RGB
    - définition d'une layer TF ad-hoc
    - utilisation de la bibliothèque skimage
  - Rescaling: [0,255] => [0,1]
    - utilisation d'une layer TF existante



Preprocessing des images: Avant/Après

original

after preprocessing

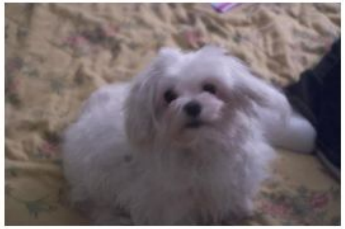
original

after preprocessing

original

after preprocessing

original

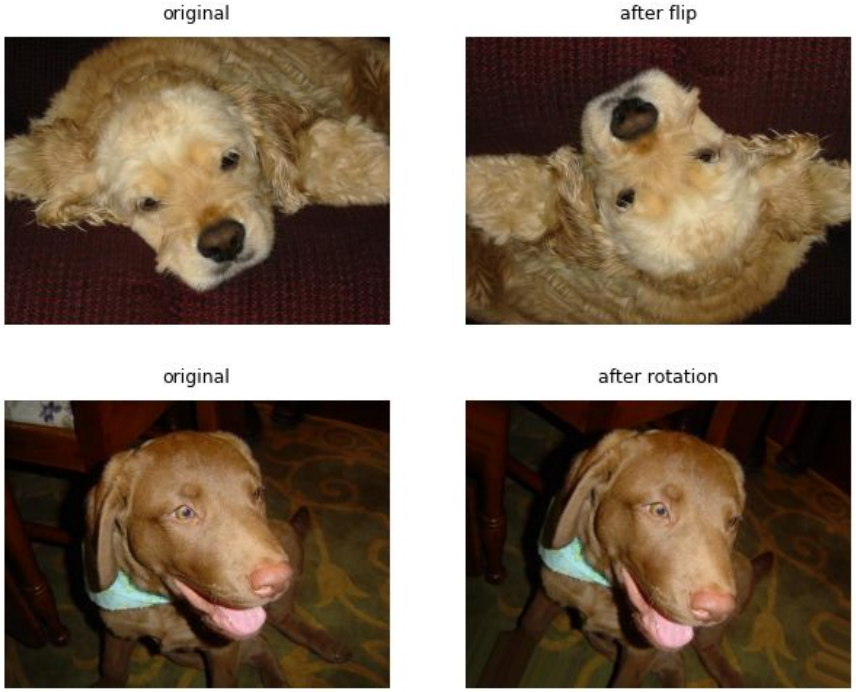


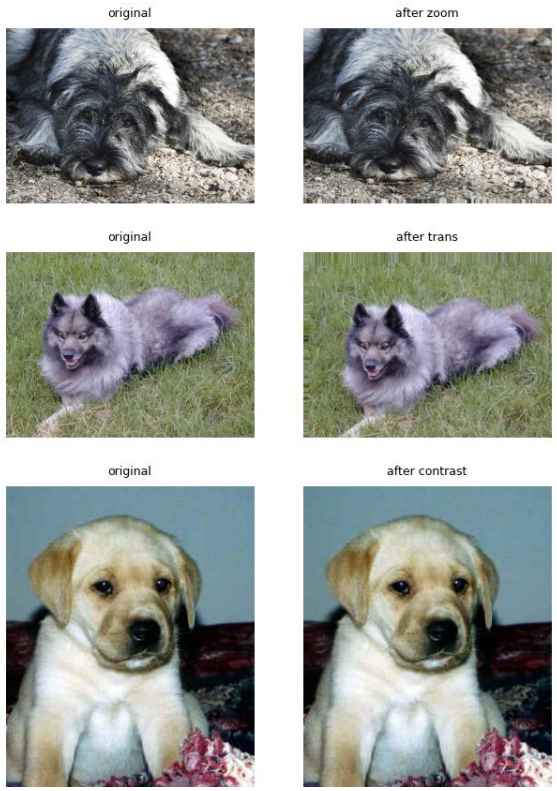
after preprocessing

Augmentation de données

- Nécessité d'avoir plus de données d'entrée (over-fitting).
- Création de nouvelles images à partir des images existantes grâce à des transformations:
  - rotation
  - flip (retournement selon x,y)
  - translations (selon x,y)
  - zoom (selon x,y)
  - modification du contraste
- Le choix des images sur lesquelles se fait la transformation est aléatoirement
- Les paramètres des transformations sont également aléatoirement

Augmentation de données: Avant/Après





Hyperparamètres et hyper-modèles

- Les poids (weights) d'un réseau de neurones appris pendant la phase d'entraînement
- D'autres paramètres sont déterminés avant la phase d'entraînement: les hyper-paramètres du modèles.
    - Une phase en amont de l'entraînement pour optimiser ces hyper-paramètres est nécessaire.
- Utilisation de l'outil "Keras Tuner" pour optimiser les hyper-paramètres du modèle.
- Hyper-paramètres définis:
    - learning rate
    - drop-out rate
    - fonction d'activation: relu, tanh
- Définition d'hyper-modèles sur la base de ces hyper-paramètres.

Hyperparamètres et hyper-modèles

```python
# hyperparameters accesssors according to mode
def rate_i(hp,mode,i,v):
  rate_name = f"rate_{str(i)}"
  if mode == "tune":
    #return  hp.Choice(name = rate_name, values = v)
    return  hp.Float(name = rate_name, min_value=v[0], max_value=v[1], sampling="linear")
  else:
    return  hp.get(name = rate_name)

def learning_rate(hp,mode):
  if mode == "tune":
    return hp.Float("lr", min_value=1e-4, max_value=1e-2, sampling="log")
  else:
    return hp.get("lr")

def activation(hp,mode,i):
  if mode == "tune":
    return hp.Choice(f"act_{i}", values = ['relu','tanh'])
  else:
    return hp.get(f"act_{i}")

# hyper model
def hyper_model_gen(hp,model_name,mode,num_classes,models):

  model = (models.get(model_name))(hp,mode,num_classes)

  if model is None:
    raise Exception(f"Unknown model {model_name}")

  return model
```

Hyper-paramètres

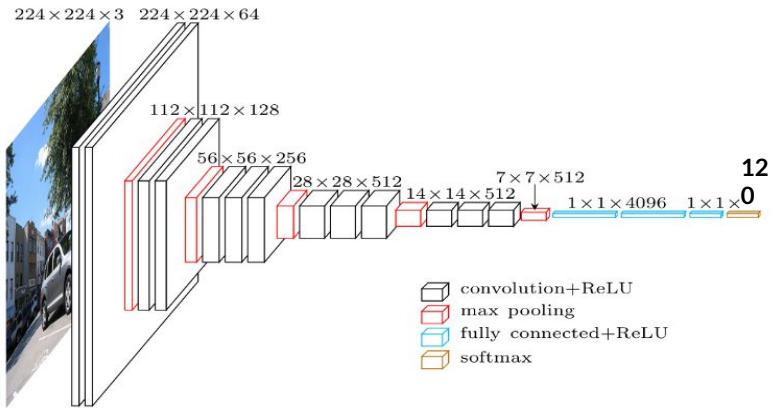
Génération des hyper-modèles sur la base de description des modèles

Hyperparamètres et hyper-modèles

```python
# hyper-model my_VGG16_1
def my_VGG16_1(hp,mode,num_classes):
  model = tf.keras.models.Sequential(
    name= "my_VGG16_1",
    layers = [
      tf.keras.Input(name="input",shape=IMG_SIZE + (3,)),
      Equalize(),
      convolution(filters=64,activation=activation(hp,mode,0)),
      convolution(filters=64,activation=activation(hp,mode,1)),
      max_pooling(),
      convolution(filters=128,activation=activation(hp,mode,2)),
      convolution(filters=128,activation=activation(hp,mode,3)),
      max_pooling(),
      convolution(filters=256,activation=activation(hp,mode,4)),
      convolution(filters=256,activation=activation(hp,mode,5)),
      convolution(filters=256,activation=activation(hp,mode,6)),
      max_pooling(),
      convolution(filters=512,activation=activation(hp,mode,7)),
      convolution(filters=512,activation=activation(hp,mode,8)),
      convolution(filters=512,activation=activation(hp,mode,9)),
      max_pooling(),
      convolution(filters=512,activation=activation(hp,mode,10)),
      convolution(filters=512,activation=activation(hp,mode,11)),
      convolution(filters=512,activation=activation(hp,mode,12)),
      max_pooling(),
      tf.keras.layers.Flatten(),
      tf.keras.layers.Dropout(rate=rate_i(hp,mode,0,[0.5,0.8])),
      hp_fully_connected(activation=activation(hp,mode,13)),
      tf.keras.layers.BatchNormalization(),
      tf.keras.layers.Dropout(rate=rate_i(hp,mode,1,[0.5,0.8])),
      hp_fully_connected(activation=activation(hp,mode,14)),
      tf.keras.layers.BatchNormalization(),
      tf.keras.layers.Dropout(rate=rate_i(hp,mode,2,[0.5,0.8])),
      soft_max(units=num_classes)]
    )

  model.compile(loss="categorical_crossentropy",
                metrics=["accuracy"],
                optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate(hp,mode))
                )

  return model
```

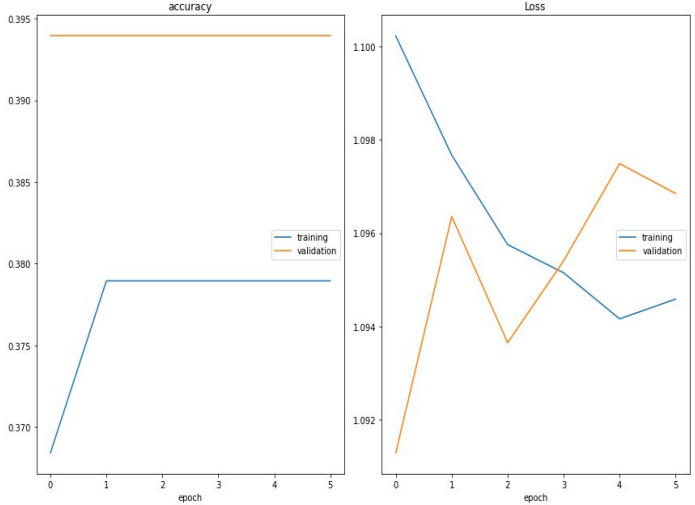
Modèles à entraîner

- Deux modèles de types VGG16 à entraîner entièrement ont été mis en place.
- Essais sur le jeu de données complet => temps de traitement rédhibitoire.
- Entraînement avec un nombre limité de données:
  - 3 races de chien (380 train, 67 val , 48 test)

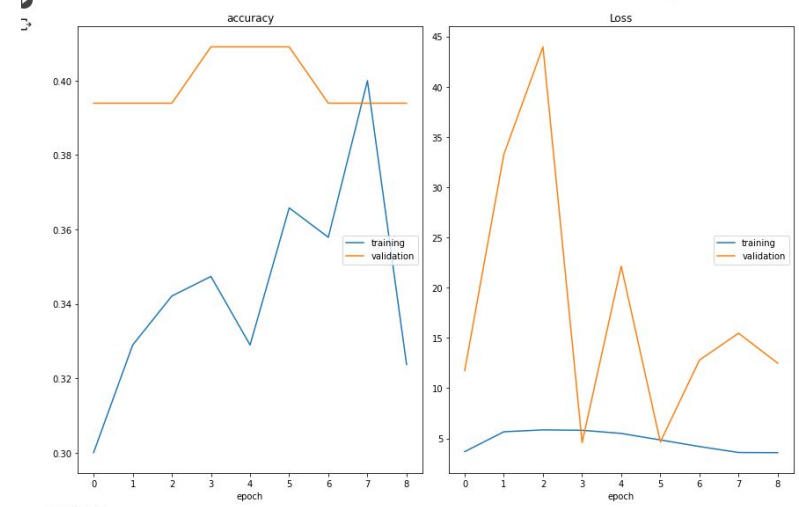


- un modèle avec un seul hyper-paramètre (learning rate) sans drop-out
- un autre avec hyper-paramètre et drop-out (qui permettent d'éviter l'over-fitting)

Modèles à entraîner: Résultats




```
accuracy
        training                (min:    0.368, max:    0.379, cur:    0.379)
        validation              (min:    0.394, max:    0.394, cur:    0.394)
Loss
        training                (min:    1.094, max:    1.100, cur:    1.095)
        validation              (min:    1.091, max:    1.097, cur:    1.097)
12/12 [==============================] - 22s 2s/step - loss: 1.0946 - accuracy: 0.3789 - val_loss: 1.0968 - val_accuracy: 0.3939
Epoch 6: early stopping

===== Load best model from: /content/drive/MyDrive/FormationML/P6/data/models/my_VGG16_0_20220713-154032_dev.h5 =====

2/2 [==============================] - 3s 2s/step - loss: 1.0952 - accuracy: 0.3750

===== EVALUATE =====

test loss, test acc: [1.0952141284942627, 0.375]

====================
```

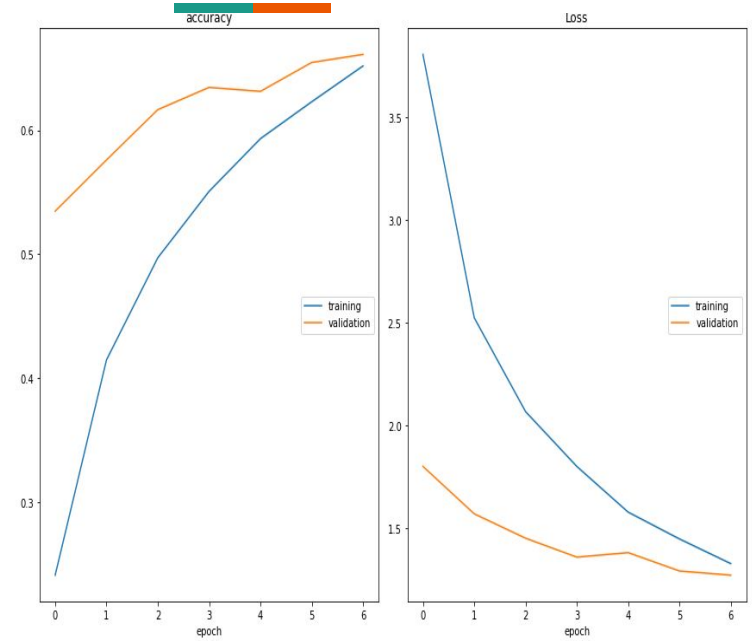



```
accuracy
        training                (min:    0.300, max:    0.400, cur:    0.324)
        validation              (min:    0.394, max:    0.409, cur:    0.394)
Loss
        training                (min:    3.579, max:    5.844, cur:    3.579)
        validation              (min:    4.573, max:   43.961, cur:   12.480)
12/12 [==============================] - 19s 2s/step - loss: 3.5791 - accuracy: 0.3237 - val_loss: 12.4797 - val_accuracy: 0.3939
Epoch 9: early stopping

===== Load best model from: /content/drive/MyDrive/FormationML/P6/data/models/my_VGG16_1_20220713-084630_dev.h5 =====

2/2 [==============================] - 3s 2s/step - loss: 6.6173 - accuracy: 0.3125

===== EVALUATE =====

test loss, test acc: [6.617256164550781, 0.3125]

====================
```

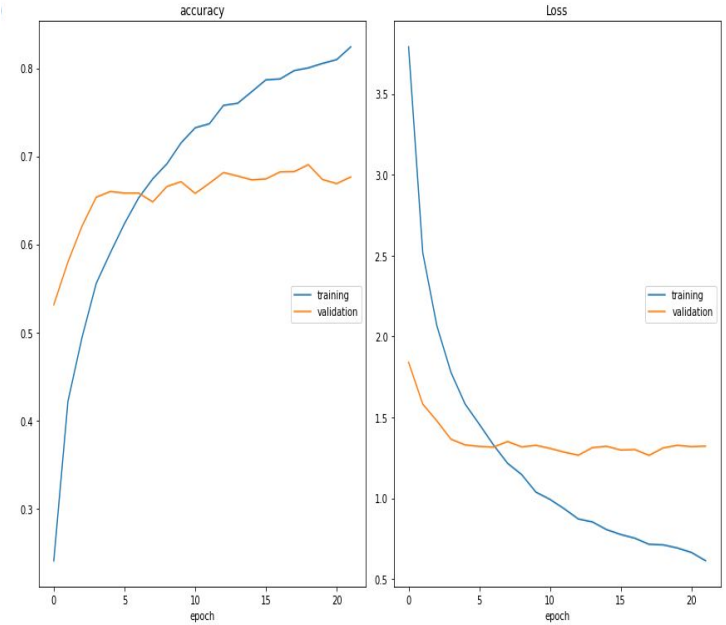
Transfert Learning

- Utilisation du même modèle VGG16 pré-entraîné sur le dataset Image-net
- Utilisation uniquement de la partie “Feature Extraction” et de la couche de pré-processing présents dans Tensor Flow.
- “Fine-tuning” de la couche dense en sortie du modèle.

Transfert Learning: Résultats




```
accuracy
        training                (min:    0.241, max:    0.652, cur:    0.652)
        validation              (min:    0.535, max:    0.661, cur:    0.661)
Loss
        training                (min:    1.327, max:    3.806, cur:    1.327)
        validation              (min:    1.270, max:    1.800, cur:    1.270)
494/494 [==============================] - 720s 1s/step - loss: 1.3265 - accuracy: 0.6518 - val_loss: 1.2703 - val_accuracy: 0.6610
Epoch 8/100
247/494 [==============>..............] - ETA: 5:18 - loss: 1.2289 - accuracy: 0.6731
```




```
accuracy
        training                (min:    0.240, max:    0.824, cur:    0.824)
        validation              (min:    0.531, max:    0.690, cur:    0.676)
Loss
        training                (min:    0.614, max:    3.790, cur:    0.614)
        validation              (min:    1.265, max:    1.841, cur:    1.322)
494/494 [==============================] - 740s 1s/step - loss: 0.6139 - accuracy: 0.8243 - val_loss: 1.3222 - val_accuracy: 0.6765
Epoch 23/100
220/494 [============>.................] - ETA: 6:08 - loss: 0.5876 - accuracy: 0.8303
```

Transfert Learning: Résultats

- Difficulté rencontrée avec le temps de traitement (déconnexion avant la fin).
- Néanmoins “accuracy” supérieure à 66% à chaque fois.
- Evaluation sur le jeu de test satisfaisant:

```
model_file = 'my_pretrained_vgg1_20220713-091942_dev.h5'
best_model= tf.keras.models.load_model(f'/content/drive/MyDrive/FormationML/P6/data/models/{model_file}',custom_objects={'Equalize':Equalize})
evaluation = best_model.evaluate(test_ds)
print("\n===== EVALUATE =====\n")
print(f"test loss, test acc: {evaluation}")
print("\n====================\n")
```

```
63/63 [==============================] - 58s 906ms/step - loss: 1.3189 - accuracy: 0.6745

===== EVALUATE =====

test loss, test acc: [1.3189197778701782, 0.6744648814201355]

====================
```

Utilisation du modèle: Résultats

```
[ ] pred_model = PredictBreed(str(P6_PATH/'models/my_pretrained_vgg1_20220712-102633_dev.h5'),str(P6_PATH/'models/class_names.save'))
    p = img_dir.path()
    predicted_breed = pred_model.predict(p)
    print(f"{p} Vs {predicted_breed}")
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/data/ops/structured_function.py:265: UserWarning: Even though the `tf.cor
  "Even though the `tf.config.experimental_run_functions_eagerly` "
/content/Images/n02094258-Norwich_terrier/n02094258_897.jpg Vs Norwich_terrier
```

```
[ ] pred_model = PredictBreed(str(P6_PATH/'models/my_pretrained_vgg1_20220712-102633_dev.h5'),str(P6_PATH/'models/class_names.save'))
    p = img_dir.path()
    predicted_breed = pred_model.predict(p)
    print(f"{p} Vs {predicted_breed}")
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/data/ops/structured_function.py:265: UserWarning: Even though the `tf.cor
  "Even though the `tf.config.experimental_run_functions_eagerly` "
/content/Images/n02086646-Blenheim_spaniel/n02086646_4133.jpg Vs Blenheim_spaniel
```

```
pred_model = PredictBreed(str(P6_PATH/'models/my_pretrained_vgg1_20220713-091942_dev.h5'),str(P6_PATH/'models/class_names.save'))
p = img_dir.path()
predicted_breed = pred_model.predict(p)
print(f"{p} Vs {predicted_breed}")
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/data/ops/structured_function.py:265: UserWarning: Even though the `tf.cor
  "Even though the `tf.config.experimental_run_functions_eagerly` "
/content/Images/n02113978-Mexican_hairless/n02113978_838.jpg Vs Mexican_hairless
```

+ Code + Text