image_classification

July 27, 2021

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import os
from PIL import Image as pil_img
```

[2]:
```
# Importer la base de données d'images
# Il s'agit ici d'un répertoire local
import pathlib
data_dir = pathlib.Path("baseDeDonnees/")

# Nombre d'images "jpg" dans la base d'images
image_count = len(list(data_dir.glob('*/*.jpg')))
print("Nombre d'images 'JPEG' :", image_count)
```

```
Nombre d'images 'JPEG' : 2003
```

[3]:
```
# Visualiser quelques fleurs de la base de données
roses = list(data_dir.glob('roses/*'))
tulips = list(data_dir.glob('tulips/*'))
```

[4]:
```
pil_img.open(str(roses[10]))
```

[4]:

[5]:
```python
pil_img.open(str(roses[1]))
```

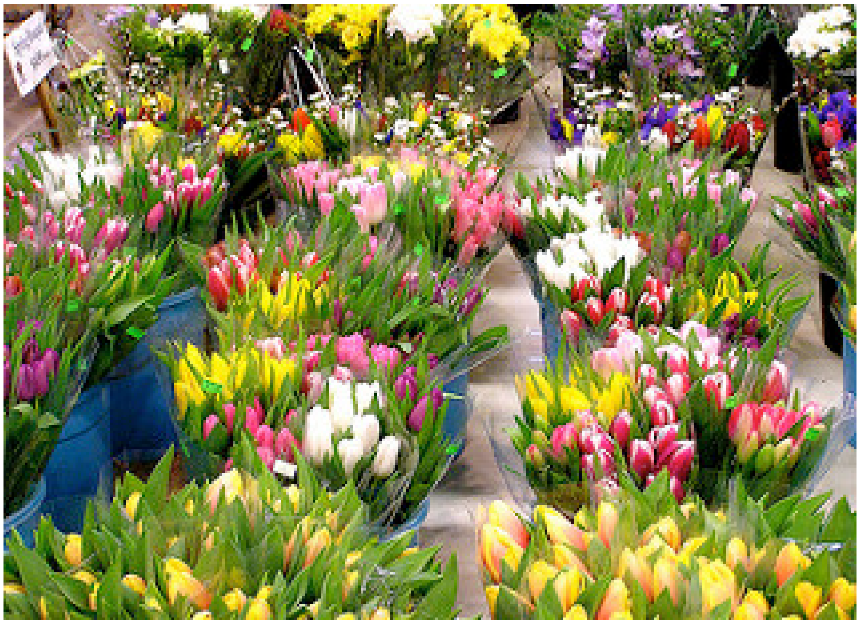
[5]:

[6]:
```python
pil_img.open(str(tulips[0]))
```

[6]:

[7]:
```python
pil_img.open(str(tulips[1]))
```

[7]:



[8]:
```
batch_size = 16
img_height = 180
img_width = 180
```

[9]:
```
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

# Création d'un modèle séquentiel
num_classes = 5

model = Sequential([
    layers.experimental.preprocessing.Rescaling(1./255,␣
 ↪input_shape=(img_height, img_width, 3)),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes)
])
```

[10]:
```
# Compilation du modèle
model.compile(optimizer='adam',
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

[11]:
```
from tensorflow import keras
# Préparation des données d'apprentissage
train_ds = keras.preprocessing.image_dataset_from_directory(data_dir,
                                                              ␣
 ↪validation_split=0.2,
                                                              ␣
 ↪subset="training",
                                                               seed=123,
                                                              ␣
 ↪image_size=(img_height, img_width),
                                                              ␣
 ↪batch_size=batch_size)

# Préparation des données de validation
val_ds = keras.preprocessing.image_dataset_from_directory(data_dir,
                                                          validation_split=0.2,
                                                          subset="validation",
                                                          seed=123,
                                                         ␣
 ↪image_size=(img_height, img_width),
                                                          batch_size=batch_size)
```

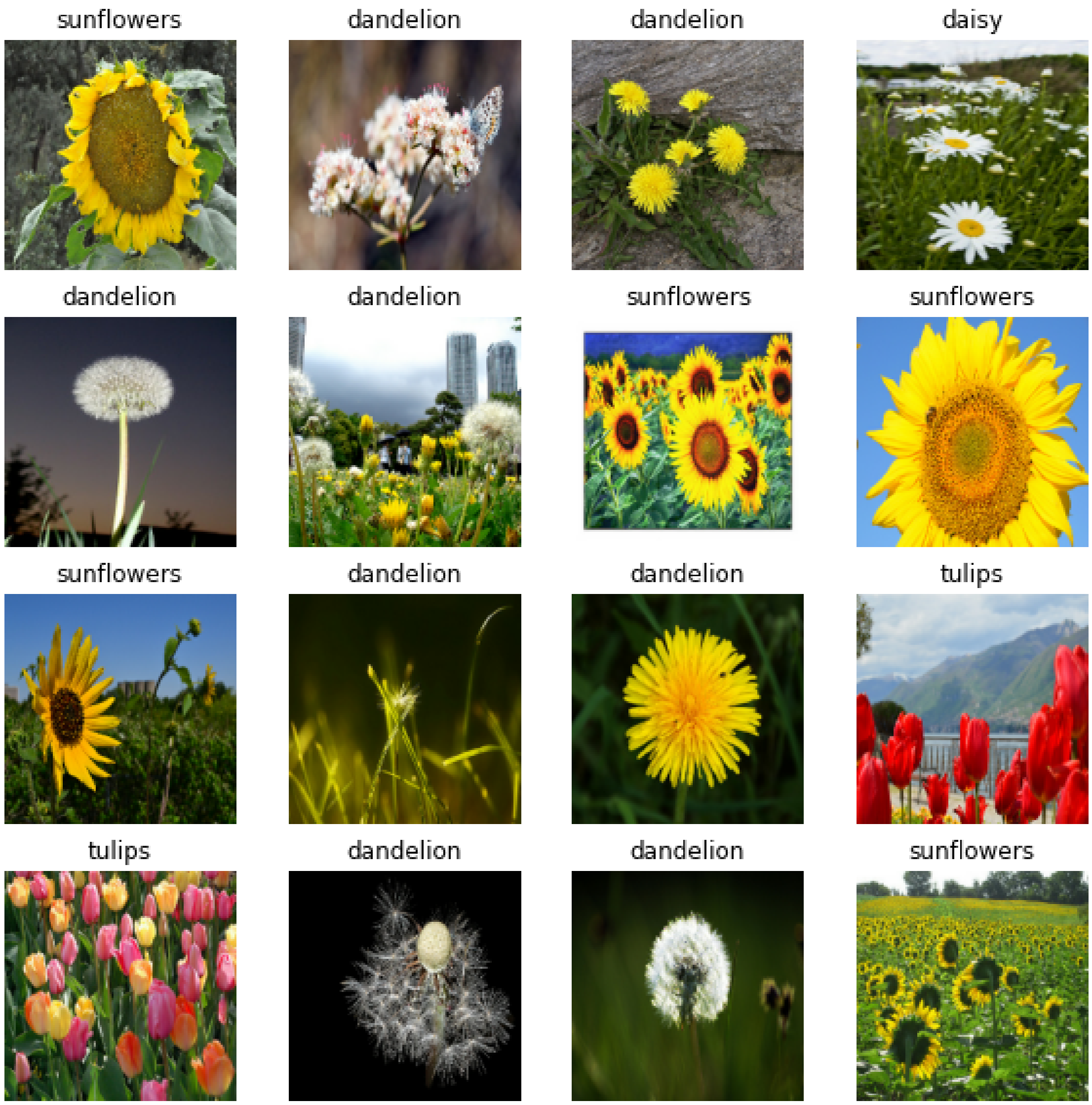
```
Found 2003 files belonging to 5 classes.
Using 1603 files for training.
Found 2003 files belonging to 5 classes.
Using 400 files for validation.
```

[12]:
```
# Afficher quelques données
import matplotlib.pyplot as plt

class_names = train_ds.class_names
print("Différentes classes :", class_names)

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(0, 16):
    ax = plt.subplot(4, 4, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```

```
Différentes classes : ['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
```



[13]:
```
image_batch, labels_batch = next(iter(train_ds))
first_image = image_batch[0]
print("Valeurs des pixels avant normalisation", np.min(first_image), np.max(first_image))

# Création d'une couche de normalisation pour normaliser les valeurs de pixels entre 0 et 1
normalization_layer = layers.experimental.preprocessing.Rescaling(1./255)
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
```

```
# Les valeurs de pixels sont désormais compris entre 0 et 1 au lieu de 0 et 255
print("Valeurs des pixels après normalisation", np.min(first_image), np.
 ↪max(first_image))
```

```
Valeurs des pixels avant normalisation 0.0 255.0
Valeurs des pixels après normalisation 0.0 1.0
```

[ ]:
```
# Entrainement du modèle
epochs=10
history = model.fit(train_ds,
                    validation_data=val_ds,
                    epochs=epochs)
```

```
Epoch 1/10
101/101 [==============================] - 41s 398ms/step - loss: 0.0837 -
accuracy: 0.9788 - val_loss: 1.9406 - val_accuracy: 0.6675
Epoch 2/10
101/101 [==============================] - 32s 310ms/step - loss: 0.0375 -
accuracy: 0.9888 - val_loss: 2.1577 - val_accuracy: 0.6275
Epoch 3/10
 81/101 [=======================>…] - ETA: 5s - loss: 0.0344 - accuracy:
0.9877
```

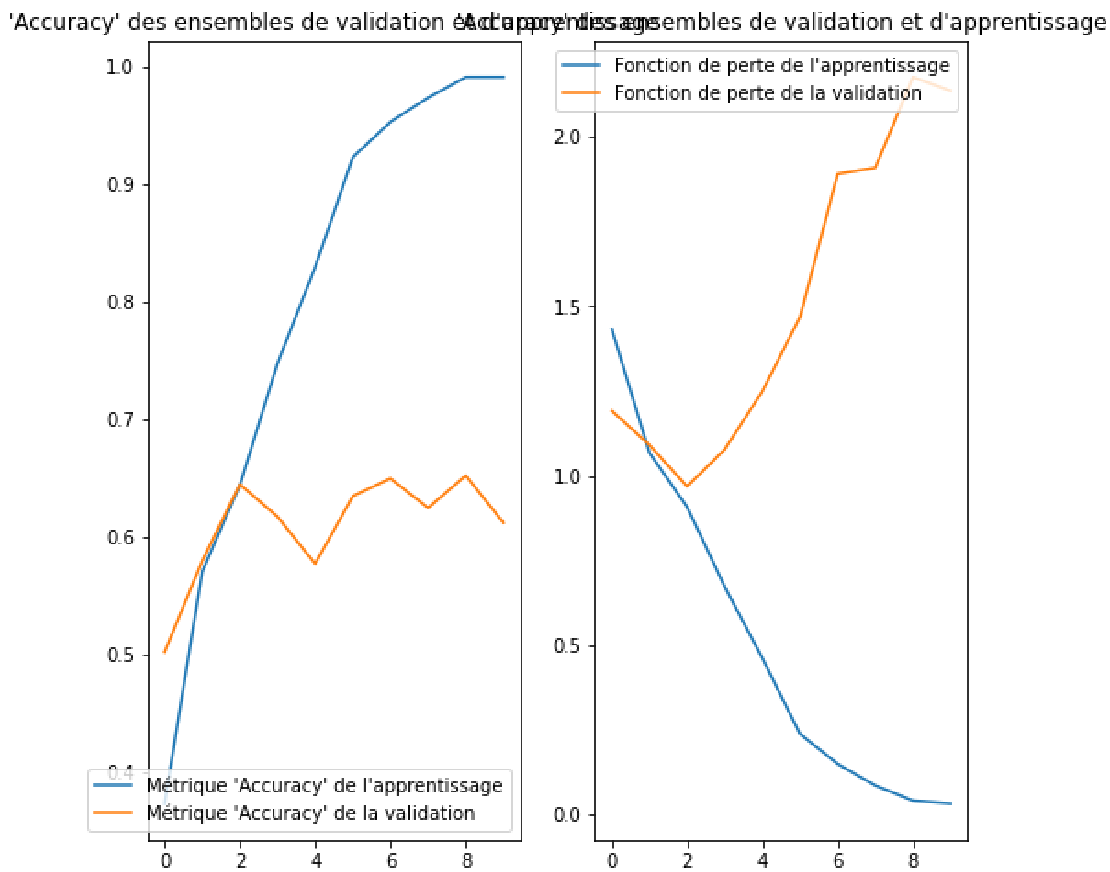
[15]:
```
# Visualisation des métriques de la phase d'appentissage
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(12, 12))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label="Métrique 'Accuracy' de l'apprentissage")
plt.plot(epochs_range, val_acc, label="Métrique 'Accuracy' de la validation")
plt.legend(loc='lower right')
plt.title("'Accuracy' des ensembles de validation et d'apprentissage")

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label="Fonction de perte de l'apprentissage")
plt.plot(epochs_range, val_loss, label="Fonction de perte de la validation")
plt.legend(loc='upper right')
plt.title("'Loss' des ensembles de validation et d'apprentissage")
plt.show()
```



[17]:

```
# Augmentation de données pour améliorer la qualité de l'apprentissage
data_augmentation = keras.Sequential(
  [
    layers.experimental.preprocessing.RandomFlip("horizontal",
                                                 input_shape=(img_height, img_width, 3)),
    layers.experimental.preprocessing.RandomRotation(0.1),
    layers.experimental.preprocessing.RandomZoom(0.1),
  ]
)

plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
  for i in range(9):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[5].numpy().astype("uint8"))
    plt.axis("off")
```

[20]:
```
# Utilisation du dropout pour réduire le sur-apprentissage
new_model = Sequential([data_augmentation,
                        layers.experimental.preprocessing.Rescaling(1./255),
                        layers.Conv2D(16, 3, padding='same', activation='relu'),
                        layers.MaxPooling2D(),
                        layers.Conv2D(32, 3, padding='same', activation='relu'),
                        layers.MaxPooling2D(),
                        layers.Conv2D(64, 3, padding='same', activation='relu'),
                        layers.MaxPooling2D(),
                        # On rajoute ici une couche de Dropout
                        layers.Dropout(0.2),
                        layers.Flatten(),
                        layers.Dense(128, activation='relu'),
```

```
                    layers.Dense(num_classes)])
```

[21]:
```
# Compilation du modèle
new_model.compile(optimizer='adam',
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

[22]:
```
# Nouvel entrainement du modèle
epochs=15
history = new_model.fit(train_ds, validation_data=val_ds,
                        epochs=epochs)
```

```
Epoch 1/15
101/101 [==============================] - 40s 392ms/step - loss: 1.4134 -
accuracy: 0.3980 - val_loss: 1.2143 - val_accuracy: 0.5100
Epoch 2/15
101/101 [==============================] - 36s 357ms/step - loss: 1.1370 -
accuracy: 0.5153 - val_loss: 1.0821 - val_accuracy: 0.5275
Epoch 3/15
101/101 [==============================] - 36s 357ms/step - loss: 1.0045 -
accuracy: 0.5933 - val_loss: 1.0007 - val_accuracy: 0.6450
Epoch 4/15
101/101 [==============================] - 36s 350ms/step - loss: 0.9582 -
accuracy: 0.6188 - val_loss: 0.8783 - val_accuracy: 0.6625
Epoch 5/15
101/101 [==============================] - 36s 357ms/step - loss: 0.8729 -
accuracy: 0.6525 - val_loss: 0.9315 - val_accuracy: 0.6725
Epoch 6/15
101/101 [==============================] - 34s 336ms/step - loss: 0.8294 -
accuracy: 0.6850 - val_loss: 0.8041 - val_accuracy: 0.6975
Epoch 7/15
101/101 [==============================] - 35s 344ms/step - loss: 0.8034 -
accuracy: 0.6956 - val_loss: 0.8394 - val_accuracy: 0.6800
Epoch 8/15
101/101 [==============================] - 35s 341ms/step - loss: 0.7592 -
accuracy: 0.7024 - val_loss: 0.7844 - val_accuracy: 0.7050
Epoch 9/15
101/101 [==============================] - 35s 342ms/step - loss: 0.7392 -
accuracy: 0.7174 - val_loss: 0.7630 - val_accuracy: 0.7050
Epoch 10/15
101/101 [==============================] - 35s 349ms/step - loss: 0.7208 -
accuracy: 0.7168 - val_loss: 0.8019 - val_accuracy: 0.7025
Epoch 11/15
101/101 [==============================] - 34s 336ms/step - loss: 0.6630 -
accuracy: 0.7386 - val_loss: 0.7855 - val_accuracy: 0.7025
Epoch 12/15
101/101 [==============================] - 35s 342ms/step - loss: 0.6132 -
```

```
accuracy: 0.7698 - val_loss: 0.7400 - val_accuracy: 0.7075
Epoch 13/15
101/101 [==============================] - 34s 335ms/step - loss: 0.5866 -
accuracy: 0.7823 - val_loss: 0.7341 - val_accuracy: 0.7275
Epoch 14/15
101/101 [==============================] - 35s 343ms/step - loss: 0.5779 -
accuracy: 0.7854 - val_loss: 0.8267 - val_accuracy: 0.7000
Epoch 15/15
101/101 [==============================] - 35s 341ms/step - loss: 0.5291 -
accuracy: 0.8141 - val_loss: 0.7623 - val_accuracy: 0.7150
```

[24]:

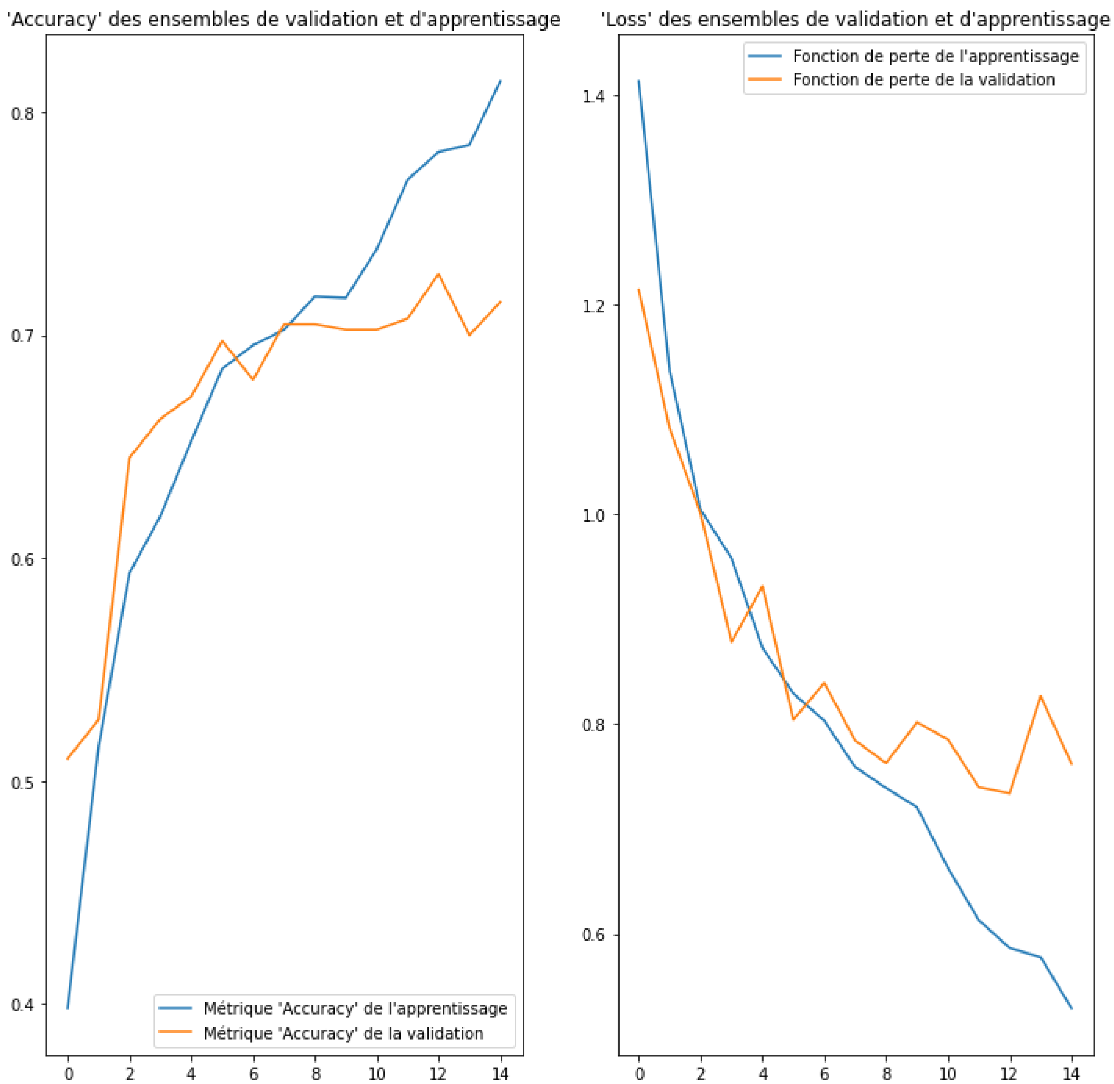
```
# Visualisation des métriques de la phase d'appentissage
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(12, 12))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label="Métrique 'Accuracy' de l'apprentissage")
plt.plot(epochs_range, val_acc, label="Métrique 'Accuracy' de la validation")
plt.legend(loc='lower right')
plt.title("'Accuracy' des ensembles de validation et d'apprentissage")

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label="Fonction de perte de l'apprentissage")
plt.plot(epochs_range, val_loss, label="Fonction de perte de la validation")
plt.legend(loc='upper right')
plt.title("'Loss' des ensembles de validation et d'apprentissage")
plt.show()
```


'Accuracy' des ensembles de validation et d'apprentissage
Métrique 'Accuracy' de l'apprentissage
Métrique 'Accuracy' de la validation
0.4
0.5
0.6
0.7
0.8
0
2
4
6
8
10
12
14
'Loss' des ensembles de validation et d'apprentissage
Fonction de perte de l'apprentissage
Fonction de perte de la validation
0.6
0.8
1.0
1.2
1.4
0
2
4
6
8
10
12
14

[ ]: