Flowers Classification

@Author: Simone Circo 2021

[3]:
```
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

[4]:
```
import pathlib
dataset_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz"
data_dir = tf.keras.utils.get_file('flower_photos', origin=dataset_url, untar=True)
data_dir = pathlib.Path(data_dir)
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz
228818944/228813984 [==============================] - 217s 1us/step
```

[8]:
```
batch_size = 32
img_height = 180
img_width = 180

train_ds = tf.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)
```

```
Found 3670 files belonging to 5 classes.
Using 2936 files for training.
```

[9]:
```
val_ds = tf.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)
```

[13]:
```
AUTOTUNE = tf.data.experimental.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

normalization_layer = layers.experimental.preprocessing.Rescaling(1./255)
```

[14]:
```
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]
# Notice the pixels values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image))
```

```
0.0 1.0
```

[15]:
```
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)
])
```

[16]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

[18]:
```
epochs=10
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/10
92/92 [==============================] - 39s 422ms/step - loss: 1.7690 - accuracy: 0.2789 - val_loss: 1.0894 - val_accuracy: 0.5490
Epoch 2/10
92/92 [==============================] - 38s 415ms/step - loss: 0.9977 - accuracy: 0.5991 - val_loss: 0.9615 - val_accuracy: 0.6063
Epoch 3/10
92/92 [==============================] - 39s 420ms/step - loss: 0.8148 - accuracy: 0.6820 - val loss: 0.9407 - val accuracy: 0.6444
92/92 [==============================] - 40s 435ms/step - loss: 0.6201 - accuracy: 0.7850 - val_loss: 0.9638 - val_accuracy: 0.6608
Epoch 5/10
92/92 [==============================] - 40s 436ms/step - loss: 0.3937 - accuracy: 0.8606 - val_loss: 1.0559 - val_accuracy: 0.6267
Epoch 6/10
92/92 [==============================] - 41s 441ms/step - loss: 0.2446 - accuracy: 0.9212 - val_loss: 1.1138 - val_accuracy: 0.6376
Epoch 7/10
92/92 [==============================] - 39s 428ms/step - loss: 0.1188 - accuracy: 0.9693 - val_loss: 1.3499 - val_accuracy: 0.6267
Epoch 8/10
92/92 [==============================] - 36s 392ms/step - loss: 0.0542 - accuracy: 0.9889 - val_loss: 1.5431 - val_accuracy: 0.6567
Epoch 9/10
92/92 [==============================] - 38s 414ms/step - loss: 0.0442 - accuracy: 0.9908 - val_loss: 1.7306 - val_accuracy: 0.6362
Epoch 10/10
92/92 [==============================] - 39s 426ms/step - loss: 0.0352 - accuracy: 0.9922 - val_loss: 1.6862 - val_accuracy: 0.6540
```

```
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=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

[20]:
```
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),
  ]
)
```

[22]:
```
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(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

[23]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

[24]:
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_1 (Sequential)    (None, 180, 180, 3)       0
_________________________________________________________________
rescaling_2 (Rescaling)      (None, 180, 180, 3)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 180, 180, 16)      448
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 90, 90, 16)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 90, 90, 32)        4640
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 45, 45, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 45, 45, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 22, 22, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 22, 22, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 30976)             0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               3965056
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 645
=================================================================
Total params: 3,989,285
Trainable params: 3,989,285


Non-trainable params: 0
_________________________________________________________________
```

[79]:
```
epochs = 15
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/15
92/92 [==============================] - 40s 438ms/step - loss: 0.4629 - accuracy: 0.8243 - val_loss: 0.6798 - val_accuracy: 0.7507
Epoch 2/15
92/92 [==============================] - 41s 443ms/step - loss: 0.4425 - accuracy: 0.8331 - val_loss: 0.6952 - val_accuracy: 0.7411
Epoch 3/15
92/92 [==============================] - 41s 446ms/step - loss: 0.4423 - accuracy: 0.8369 - val_loss: 0.7529 - val_accuracy: 0.7262
Epoch 4/15
92/92 [==============================] - 41s 450ms/step - loss: 0.4346 - accuracy: 0.8375 - val_loss: 0.7002 - val_accuracy: 0.7357
Epoch 5/15
92/92 [==============================] - 41s 451ms/step - loss: 0.3886 - accuracy: 0.8498 - val_loss: 0.7681 - val_accuracy: 0.7193
Epoch 6/15
92/92 [==============================] - 41s 449ms/step - loss: 0.3631 - accuracy: 0.8665 - val_loss: 0.7345 - val_accuracy: 0.7371
Epoch 7/15
92/92 [==============================] - 41s 449ms/step - loss: 0.3555 - accuracy: 0.8706 - val_loss: 0.7612 - val_accuracy: 0.7371
Epoch 8/15
92/92 [==============================] - 41s 450ms/step - loss: 0.3195 - accuracy: 0.8828 - val_loss: 0.7311 - val_accuracy: 0.7411
Epoch 9/15
92/92 [==============================] - 42s 453ms/step - loss: 0.3280 - accuracy: 0.8757 - val_loss: 0.7830 - val_accuracy: 0.7466
Epoch 10/15
92/92 [==============================] - 42s 453ms/step - loss: 0.3138 - accuracy: 0.8835 - val_loss: 0.7767 - val_accuracy: 0.7411
Epoch 11/15
92/92 [==============================] - 41s 450ms/step - loss: 0.2906 - accuracy: 0.8971 - val_loss: 0.7903 - val_accuracy: 0.7561
Epoch 12/15
92/92 [==============================] - 41s 451ms/step - loss: 0.2730 - accuracy: 0.9040 - val_loss: 0.7906 - val_accuracy: 0.7561
Epoch 13/15
92/92 [==============================] - 41s 450ms/step - loss: 0.2530 - accuracy: 0.9046 - val_loss: 0.8133 - val_accuracy: 0.7534
Epoch 14/15
92/92 [==============================] - 41s 451ms/step - loss: 0.2245 - accuracy: 0.9220 - val_loss: 0.8815 - val_accuracy: 0.7670
Epoch 15/15
92/92 [==============================] - 41s 449ms/step - loss: 0.2505 - accuracy: 0.9060 - val_loss: 0.7802 - val_accuracy: 0.7738
```

[80]:
```
sunflower_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/592px-Red_sunflower.jpg"
sunflower_path = tf.keras.utils.get_file('Red_sunflower2', origin=sunflower_url)
#sunflower_url = "https://lh3.googleusercontent.com/hFUHtOCNPSQ81ymeO6Fg3pfOqqtGv_yFF4qNs1omjbBJ-FS5AcBA-kgGc7rXKvvbvqm0rZX9HQ=w640-h400-e365-rj-sc0x00ffffff"

#sunflower_path = tf.keras.utils.get_file('test', origin=sunflower_url)

img = keras.preprocessing.image.load_img(
     sunflower_path , target_size=(img_height, img_width)
)
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
   .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

```
This image most likely belongs to sunflowers with a 99.99 percent confidence.
```