Práctica

Transfer Learning 2

Vamos a adaptar mediante un modelo pre-entrenado de *MobileNet*. Este modelo permite identificar muchas categorías de imágenes, y existen distintas versiones publicadas en diversos repositorios online, como por ejemplo en Kaggle en esta dirección: https://www.kaggle.com/models/google/mobilenet-v2

Además de eso, ya sabemos que Keras incorpora modelos de transfer learning, como su propio modelo de MobileNet que también podemos utilizar. Lo tenemos disponible dentro del paquete keras.applications. En este caso usaremos la versión MobileNetV2.

En primer lugar, vamos a utilizar directamente el modelo para que catalogue un conjunto de imágenes, viendo los errores que comete. Después desacoplaremos la capa *fully connected* del final para añadir otra propia, que entrenaremos específicamente para el nuevo objetivo, viendo cómo se mejoran los resultados. Usaremos las imágenes de comida de prueba proporcionadas en el fichero "ejemplos…".

1. Prueba del modelo sin re-entrenamiento

Como decíamos, vamos a probar directamente el modelo para ver cómo cataloga las imágenes de prueba. Construimos un pequeño programa que cargue el modelo y procese una a una las imágenes:

```
import os
import numpy as np
import PIL.Image as Image
import keras
from keras.applications import MobileNetV2
from keras.models import Sequential

tam_imagen = 224

modelo1 = Sequential([
    MobileNetV2(input_shape=(tam_imagen, tam_imagen, 3), weights='imagenet')
])

# Suponiendo que tenemos las imágenes en la subcarpeta "prueba"
for nombre_archivo in os.listdir('.\\prueba'):
    ruta_imagen = os.path.join('.\\prueba', nombre_archivo)
    if ruta_imagen.endswith(('.jpg', '.jpeg', '.png')):
        imagen = Image.open(ruta_imagen).resize((tam_imagen, tam_imagen))
        imagen = np.array(imagen) / 255
        resultado = modelo1.predict(imagen[np.newaxis, ...])
        codigo = np.argmax(resultado[0])
        fichero_etiquetas = keras.utils.get_file('ImageNetLabels.txt',
            'https://storage.googleapis.com/download.tensorflow.org/data/ImageNetLabels.txt')
        etiquetas = np.array(open(fichero_etiquetas).read().splitlines())
        print(nombre_archivo, ":", etiquetas[codigo])
```

Si ejecutamos el ejemplo, podemos ver que la clasificación es equivocada, y se asignan etiquetas que no se corresponden realmente con lo que hay en las imágenes.

2. Re-entrenando el modelo

Ahora utilizaremos el mismo modelo de una forma diferente: en lugar de cargar todo el modelo cargaremos las capas convolucionales que sirven como extractor de características, desacoplando las últimas capas *fully connected* para incorporar las nuestras propias. Así, entrenaremos sólo esta última parte de la red, dejando fijo el resto. El entrenamiento será más rápido, y la complejidad del modelo pre-entrenado que usamos hará que la identificación de las imágenes sea mejor. Para el entrenamiento usaremos el dataset de imágenes de comidas que ya hemos utilizado en un ejercicio anterior, con multitud de imágenes de las tres categorías que nos interesan (donuts, nachos y guacamole).

Así quedaría nuestro nuevo modelo:

```
from keras.layers import Dense, GlobalAveragePooling2D
from keras.layers import Rescaling, RandomFlip, RandomRotation, \
    RandomZoom, RandomTranslation
from keras.utils import image_dataset_from_directory

...

batch_size = 32
carpeta_train = '.\\imagenes_comidas\\train'
carpeta_test = '.\\imagenes_comidas\\test'
clases = ['donuts', 'guacamole', 'nachos']

# Preprocesamiento de imágenes de entrada

train_dataset = image_dataset_from_directory(
    carpeta_train,
    image_size=(tam_imagen, tam_imagen),
    batch_size=batch_size,
    labels='inferred',
    label_mode='categorical',
    class_names=clases
)

test_dataset = image_dataset_from_directory(
    carpeta_test,
    image_size=(tam_imagen, tam_imagen),
    batch_size=batch_size,
    labels='inferred',
    label_mode='categorical',
    class_names=clases
)

# Etapas de aumento de datos y escalado

data_augmentation = Sequential([
    RandomFlip("horizontal"),
```

```
    RandomRotation(0.2),
    RandomZoom(0.1),
    RandomTranslation(0.1, 0.1)
])

normalizacion = Rescaling(1./255)

train_dataset = train_dataset.map(lambda x, y:
(data_augmentation(normalizacion(x)), y))
test_dataset = test_dataset.map(lambda x, y: (normalizacion(x), y))

# Construcción del modelo

extractor = MobileNetV2(input_shape=(tam_imagen, tam_imagen, 3),
    include_top=False, weights='imagenet')
extractor.trainable = False

modelo2 = Sequential([
    extractor,
    GlobalAveragePooling2D(),
    Dense(128, activation='relu'),
    # Crearemos una red para identificar 3 tipos de comida
    Dense(3, activation='softmax')
    ]
)
modelo2.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
modelo2.fit(train_dataset, validation_data=test_dataset, epochs=5)
```