

¡Por supuesto! Vamos a desglosar este código línea por línea en español. Este código está diseñado para configurar y entrenar un modelo de clasificación de imágenes utilizando aprendizaje por transferencia con TensorFlow y Keras, utilizando redes preentrenadas como MobileNetV2:

## Importaciones y Configuración Inicial

python

```
import os, sys, math  
import numpy as np  
from matplotlib import pyplot as plt  
import tensorflow as tf  
print("Tensorflow version " + tf.__version__)
```

- **Importaciones:** Se importan las bibliotecas necesarias de Python y TensorFlow para cálculos, visualización y funcionalidades de aprendizaje profundo.
- **Print:** Muestra la versión de TensorFlow que está en uso.

python

```
AUTOTUNE = tf.data.AUTOTUNE
```

- **AUTOTUNE:** Se utiliza para la autooptimización del rendimiento de la carga de datos del conjunto de datos. Permite que TensorFlow decida automáticamente el número óptimo de hilos para las operaciones en paralelo.

python

```
GCS_PATTERN = 'gs://flowers-public/tfrecords-jpeg-192x192-2/*.tfrec'
```

```
IMAGE_SIZE = [192, 192]
```

- **GCS\_PATTERN:** Especifica el patrón de la ruta de acceso en Google Cloud Storage para acceder a los archivos TFRecord que contienen los datos de las imágenes.
- **IMAGE\_SIZE:** Especifica el tamaño objetivo de las imágenes que se van a procesar (192x192 píxeles).

python

```
BATCH_SIZE = 64 # 128 también funciona en GPU pero se acerca mucho al límite de memoria de la GPU de Colab
```

EPOCHS = 10

- **BATCH\_SIZE:** El número de imágenes procesadas en paralelo durante cada paso de entrenamiento.
- **EPOCHS:** Número de veces que el modelo será entrenado sobre todo el conjunto de datos.

python

VALIDATION\_SPLIT = 0.19

CLASSES = ['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips'] # no cambiar, corresponde con las etiquetas en los datos (nombres de las carpetas)

- **VALIDATION\_SPLIT:** La fracción del conjunto de datos reservada para validación.
- **CLASSES:** Lista de clases de flores que se están clasificando.

## Manejo de Conjuntos de Datos

python

```
filenames = tf.io.gfile.glob(GCS_PATTERN)
split = int(len(filenames) * VALIDATION_SPLIT)
training_filenames = filenames[split:]
validation_filenames = filenames[:split]
```

- **filenames:** Recopila todos los nombres de archivos TFRecord que coinciden con el patrón especificado.
- **split:** Calcula el índice para dividir los nombres de los archivos en conjuntos de entrenamiento y validación basados en VALIDATION\_SPLIT.
- **training\_filenames, validation\_filenames:** Divide los nombres de archivos del conjunto de datos para entrenamiento y validación según el índice calculado.

python

```
print("Pattern matches {} data files. Splitting dataset into {} training files and {} validation files".format(len(filenames), len(training_filenames), len(validation_filenames)))
```

- **Mensaje de impresión:** Muestra el número de archivos de datos y cómo se dividen entre los conjuntos de datos de entrenamiento y validación.

```
python
```

```
validation_steps = int(3670 // len(filenames) * len(validation_filenames)) // BATCH_SIZE  
steps_per_epoch = int(3670 // len(filenames) * len(training_filenames)) // BATCH_SIZE
```

- **validation\_steps, steps\_per\_epoch:** Calcula el número de pasos por lote para cada época y ejecución de validación. Esto maneja la iteración sobre el conjunto de datos dado el tamaño del lote y la división.

## Utilidades de Visualización y Carga de Datos

Estas funciones son códigos de utilidad para visualización de datos, visualización y conversión.

```
python
```

```
def dataset_to_numpy_util(dataset, N):
```

```
...
```

```
def title_from_label_and_target(label, correct_label):
```

```
...
```

```
def display_one_flower(image, title, subplot, red=False):
```

```
...
```

```
def display_9_images_from_dataset(dataset):
```

```
...
```

```
def display_9_images_with_predictions(images, predictions, labels):
```

```
...
```

```
def display_training_curves(training, validation, title, subplot):
```

```
...
```

- Las funciones `dataset_to_numpy_util`, `display_one_flower`, `display_9_images_from_dataset` y `display_9_images_with_predictions` se utilizan para visualizar imágenes y predicciones del modelo.
- `display_training_curves` visualiza métricas de entrenamiento y validación a lo largo de las épocas.

## Manejo de TFRecord y Conjunto de Datos (continuación)

python

```
def read_tfrecord(example):
```

...

- **read\_tfrecord**: Esta función es responsable de leer y decodificar cada ejemplo del TFRecord. Define y extrae características (como las imágenes y sus clases) y normaliza las imágenes a un rango [0, 1].

python

```
def load_dataset(filenames):
```

...

- **load\_dataset**: Usa las funciones de TensorFlow para leer múltiples archivos TFRecord en paralelo y aplicando la función read\_tfrecord para procesar cada ejemplo. Se configura para una lectura óptima y el procesamiento mediante la opción de no determinismo para mejoras en la ejecución.

python

```
def get_batched_dataset(filenames, train=False):
```

...

- **get\_batched\_dataset**: Carga los datos usando load\_dataset, los almacena en caché para mejorar la eficiencia, y forma lotes (batch) listo para entrenamiento o validación. Si train=True, los datos se repiten para entrenamiento continuo. Prefetching está habilitado para realizar cargas en segundo plano, optimizando la gestión de datos durante el entrenamiento.

## Configuración del Modelo

python

```
# Instanciar los conjuntos de datos
```

```
training_dataset = get_batched_dataset(training_filenames, train=True)
```

```
validation_dataset = get_batched_dataset(validation_filenames, train=False)
```

- **training\_dataset, validation\_dataset:** Ejemplifica cómo se utilizan las funciones de obtención de datos para configurar flujos de datos para el entrenamiento y la validación usando archivos previamente divididos.

## Configuración del Modelo de Transferencia de Aprendizaje

python

```
pretrained_model = tf.keras.applications.MobileNetV2(input_shape=[*IMAGE_SIZE, 3],  
include_top=False)
```

```
pretrained_model.trainable = False
```

- **pretrained\_model:** Utiliza un modelo preentrenado (aquí MobileNetV2) omitiendo su capa superior (include\_top=False) para que se adapte al nuevo conjunto de clases objetivo. El modelo se congela para no reentrenar sus parámetros.

## Implementación del Modelo Secuencial

python

```
model = tf.keras.Sequential([  
  
    pretrained_model,  
  
    tf.keras.layers.Flatten(),  
  
    tf.keras.layers.Dense(5, activation='softmax')  
  
])
```

- **Modelo Secuencial:** Construye un nuevo modelo que utiliza las características del modelo preentrenado, seguido de:
  - **Flatten:** Convierte la salida tridimensional del modelo preentrenado en un vector 1D.
  - **Dense con softmax:** Genera probabilidades de clase finales con 5 clases de flores.

## Compilación y Entrenamiento del Modelo

python

```
model.compile(  
    optimizer='adam',
```

```
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
)
```

- **Compilación:** Configura el modelo con el optimizador Adam para actualización eficiente de parámetros y la pérdida sparse\_categorical\_crossentropy para clasificación multiclas.

```
python
model.summary()
```

```
history = model.fit(training_dataset, steps_per_epoch=steps_per_epoch,
epochs=EPOCHS,
                    validation_data=validation_dataset, validation_steps=validation_steps)

• Resumen del modelo: Muestra los detalles de cada capa dentro del modelo compilado.

• Entrenamiento: Ejecuta el modelo usando datos de entrenamiento, validando en cada época.
```

## Evaluación y Visualización

```
python
print(history.history.keys())

display_training_curves(history.history['accuracy'], history.history['val_accuracy'],
'accuracy', 211)

display_training_curves(history.history['loss'], history.history['val_loss'], 'loss', 212)
```

- **Mostrar Curvas:** Visualiza la precisión y la pérdida para datos de entrenamiento y validación a través de un gráfico.

## Predicciones del Modelo

```
python
flowers, labels =
dataset_to_numpy_util(load_dataset(validation_filenames).skip(np.random.randint(300)),
9)
```

```
predictions = model.predict(flowers, steps=1)
```

- **Predicciones:** Muestra cómo el modelo genera predicciones sobre un subconjunto del conjunto de validación.

Este código ejemplifica el flujo de trabajo completo desde el procesamiento y carga de datos, pasando por la preparación del modelo para el entrenamiento, hasta el ajuste de un modelo de transferencia de aprendizaje de manera efectiva.