Actividad Semana 2

Resolución de un problema usando modelos pre-entrenados

Daniel González
Semana 2: Modelos Pre-entrenados
IEBS

```
import tensorflow as tf
import numpy as np
import pandas as pd

# Para mostrar gráficas
import matplotlib.pyplot as plt
%matplotlib inline

# Anaconda fixing problem
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'

# Establecemos una semilla para numpy y tensorflow para poder reproducir la ejecución y los resultados
seed = 101
np.random.seed(seed)
tf.random.set_seed(seed)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

En esta actividad vamos a seguir trabajando con modelo pre-entrenados para coger práctica y familiarizarnos con ellos. En especial vamos a familiarizarnos con la técnica de *fine-tuning* vista en clase.

Este notebook será el inicio del caso práctico, por lo que el trabajo que realicéis en esta tarea os servirá para completar el caso práctico del módulo.

Flickr Style dataset

Este dataset es el que hemos visto en la clase anterior y con el que trabajaremos en el caso práctico. Para refrescarlo, es un dataset que contiene imágenes en color donde queremos clasificar cada imagen según el estilo fotográfico al que pertenece.

El dataset de de imágenes Flickr Style tiene las siguintes características:

- Imágenes de 5 tipos de estilo: Detailed, Pastel, Melancholy, Noir y HDR.
- Imágenes en color, es decir, cada pixel tiene 3 valores entre 0 y 255, esos valores corresponden a los valores de RGB (Red, Green, Blue).
- Imágenes de diferentes tamaños, por lo que tendremos que redimensionarlas al mismo tamaño todas antes de usarlas en nuestro modelo.
- 2.000 imágenes en total para el entrenamiento y para el test.

Imporante!!! Solo ejecutar esta celda una sola vez para descargar los datos, no ejecutarlas si ya tenéis los datos descargados

```
!wget 'https://www.dropbox.com/s/ln92e9givhgzugr/flickr_style.zip?dl=0' -O flickr_style.zip
!unzip -q flickr_style.zip
!mkdir data
!mv flickr_style data/
```

```
--2025-01-05 19:40:46--  https://www.dropbox.com/s/ln92e9givhgzugr/flickr_style.zip?dl=0
Resolving www.dropbox.com (www.dropbox.com)... 162.125.2.18, 2620:100:6017:18::a27d:212
Connecting to www.dropbox.com (www.dropbox.com)|162.125.2.18|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://www.dropbox.com/scl/fi/7fp1zo061c41gqo0mjpxt/flickr_style.zip?rlkey=fsku7a85n784e691sps5wtv7o&dl=0 [following]
```

```
--2025-01-05 19:40:46--  https://www.dropbox.com/scl/fi/7fp1zo061c41gqo0mjpxt/flickr_style.zip?rlkey=fsku7a85n784e691sps5wtv7o&dl=0
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uc6ba918dce866c7c3dd29f1599c.dl.dropboxusercontent.com/cd/0/inline/Chrcf6eLpfseTDSOqOSeQICUih4xc5ByTd_VoRU60oZoLe
--2025-01-05 19:40:46--  https://uc6ba918dce866c7c3dd29f1599c.dl.dropboxusercontent.com/cd/0/inline/Chrcf6eLpfseTDSOqOSeQICUih4xc5By
Resolving uc6ba918dce866c7c3dd29f1599c.dl.dropboxusercontent.com (uc6ba918dce866c7c3dd29f1599c.dl.dropboxusercontent.com)... 162.125
Connecting to uc6ba918dce866c7c3dd29f1599c.dl.dropboxusercontent.com (uc6ba918dce866c7c3dd29f1599c.dl.dropboxusercontent.com)|162.12
HTTP request sent, awaiting response... 302 Found
Location: /cd/0/inline2/ChrLTIV53NlT6XqWLM41jjU8xDWAn4Ho1FJQ3YMbkGBIZM_gmPvJJWgYvNh9WA6lj0tAFsDX8llYwXsXBIy3kJ8hAQyYTJChIifFESGpw4Dc
--2025-01-05 19:40:47--  https://uc6ba918dce866c7c3dd29f1599c.dl.dropboxusercontent.com/cd/0/inline2/ChrLTIV53NlT6XqWLM41jjU8xDWAn4H
Reusing existing connection to uc6ba918dce866c7c3dd29f1599c.dl.dropboxusercontent.com:443.
HTTP request sent, awaiting response... 200 OK
Length: 174599535 (167M) [application/zip]
Saving to: 'flickr_style.zip'

flickr_style.zip    100%[===================>] 166.51M  82.3MB/s    in 2.0s

2025-01-05 19:40:50 (82.3 MB/s) - 'flickr_style.zip' saved [174599535/174599535]
```

Vamos a cargar los datos de las labels y los datos de las rutas de las imágenes:

```
"""# obtenemos el nombre de las primeras etiquetas seleccionadas
style_label_file = 'data/flickr_style/style_names.txt'
style_labels = list(np.loadtxt(style_label_file, str, delimiter='\n'))
style_labels"""
```

```
'# obtenemos el nombre de las primeras etiquetas seleccionadas\nstyle_label_file = 'data/flickr_style/style_names.txt'\nstyle_label
s = list(np.loadtxt(style_label_file, str, delimiter='\n'))\nstyle_labels'
```

```
# Abrimos el archivo y cargamos las etiquetas línea por línea ( este no da error el original si)
with open('data/flickr_style/style_names.txt', 'r') as file:
    style_labels = [line.strip() for line in file.readlines()]

style_labels
```

```
['Detailed', 'Pastel', 'Melancholy', 'Noir', 'HDR']
```

```
# cargamos los datos de train
train_frame = pd.read_csv('data/flickr_style/train.txt', sep=" ", header=None)
train_frame.columns = ['files','labels_idx']
train_frame['labels'] = train_frame['labels_idx'].map({i:j for i,j in enumerate(style_labels)})

train_frame.head()
```

	files	labels_idx	labels
0	data/flickr_style/images/2216312257_2ba4af8439...	3	Noir
1	data/flickr_style/images/1247783411_4b3332a10f...	3	Noir
2	data/flickr_style/images/12981126664_676c39228...	2	Melancholy
3	data/flickr_style/images/1184077873_911026e6ae...	3	Noir
4	data/flickr_style/images/8947602754_24140f40c5...	1	Pastel

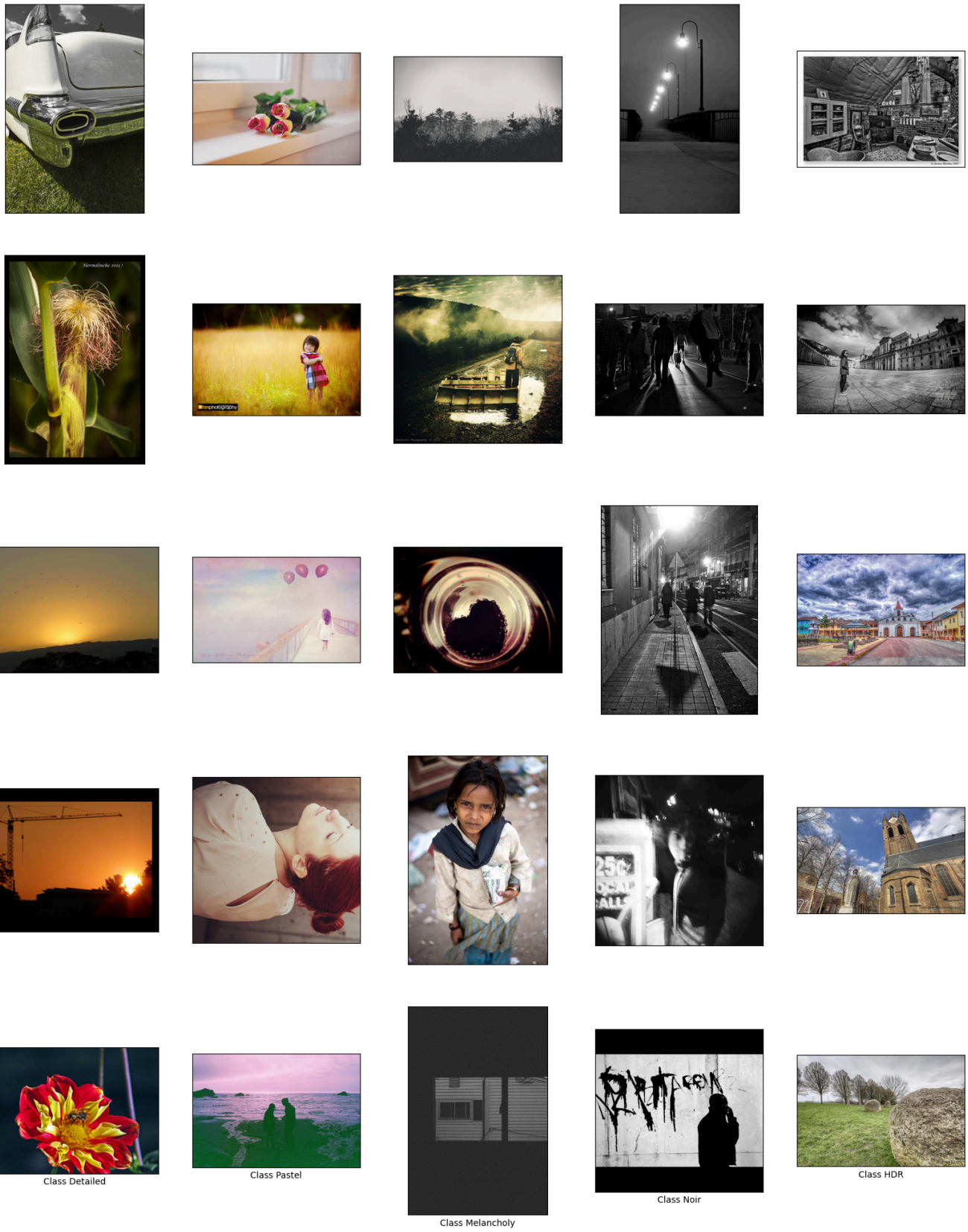
Pasos siguientes: Generar código con train_frame | Ver gráficos recomendados | New interactive sheet

```
# cargamos los datos de test
test_frame = pd.read_csv('data/flickr_style/test.txt', sep=" ", header=None)
test_frame.columns = ['files','labels_idx']
test_frame['labels'] = test_frame['labels_idx'].map({i:j for i,j in enumerate(style_labels)})

test_frame.head()
```

	files	labels_idx	labels
0	data/flickr_style/images/13059448154_d5ddf02da...	1	Pastel
1	data/flickr_style/images/13091035063_b32ef5213...	2	Melancholy
2	data/flickr_style/images/13230679494_88e5182c8...	4	HDR
3	data/flickr_style/images/12751269395_64f990535...	2	Melancholy
4	data/flickr_style/images/13285760404_05e737a5d...	2	Melancholy

Pasos siguientes: Generar código con test_frame | Ver gráficos recomendados | New interactive sheet

```
# Mostramos 5 imágenes de cada clase.
plot_n_images = 5
fig = plt.figure(figsize=(20, 25))

np.random.seed(1000)
for i in range(0,5):
    select_frame = train_frame[train_frame['labels_idx']==i]
    for j in range(0,plot_n_images):
        aux_index = np.random.choice(select_frame.index)
        fig_i=fig.add_subplot(plot_n_images,5,j*5+i+1)
        fig_i.imshow(plt.imread(train_frame['files'][aux_index]))

        fig_i.set_xticks(())
        fig_i.set_yticks(())

    fig_i.set_xlabel('Class %s' % style_labels[i])
```


Class Detailed
Class Pastel
Class Melancholy
Class Noir
Class HDR

Ejercicio 1: preprocesamiento de las imágenes

Como hemos visto en clase, para utilizar las imágenes usando un modelo pre-entrenado es necesario realizar una transformación sobre las imágenes que vamos a utilizar, es decir, tenemos que:

1. Cargar las imágenes
2. Redimensionarlas
3. Usar la función de `preprocess_input` del modelo pre-entrenado que vamos a utilizar.

En este problema vamos a utilizar otro modelo pre-entrado entre los que están disponibles en `tf.keras`, esta vez vamos a usar el modelo de *InceptionV3*, um modelo muy popular y usando frecuentemente. Podéis ver toda la información de este modelo aquí. Este modelo utilizar un tamaño de imagen de ***(299, 299, 3)***, por lo que tendrás que usar este tamaño en el redimensionamiento.

Como hemos hecho en clase, define una función que se llame `load_img_inceptionv3(img_path)` a la cual le pasamos la ruta donde está alojada una imagen y le realiza todas las transformación necesarias para poder se utilizada luego en el proceso de entrenamiento.

```
def load_img_inceptionv3(img_path):
    # Completar
    from tensorflow.keras.applications.inception_v3 import preprocess_input
    from tensorflow.keras.preprocessing.image import load_img, img_to_array
    # Cargamos la imagen desde la ruta especificada
    img = load_img(img_path, target_size=(299, 299))  # Redimensiona a (299, 299)
    # Conviertimos la imagen en un array numpy
    img_array = img_to_array(img)
    # Añadimos una dimensión para que sea compatible con el modelo
    img_array = np.expand_dims(img_array, axis=0)
    # Preprocesamos la imagen utilizando la función preprocess_input
    img_array = preprocess_input(img_array)
    return img_array
```

Test: Puedes probar esta función con el siguiente test, cuando la tengas definida ejecuta la siguiente celda y te debería dar como resultado:

```
 -0.5529412
```

```
img = load_img_inceptionv3('data/flickr_style/images/2216312257_2ba4af8439.jpg')
img[0,0,0]
```

```
array([-0.5529412, -0.5529412, -0.5372549], dtype=float32)
```

Ejercicio 2: aplicar el preprocesamiento a todas las imágenes

Una vez tenemos definida nuestra función de para transformar las imágenes, aplíca la transformación tanto al conjunto de train como al conjunto de test como hemos visto en clase.

```
# cargamos las imágenes ya transformadas
x_train =  np.array([load_img_inceptionv3(path) for path in train_frame['files']])
x_test = np.array([load_img_inceptionv3(path) for path in test_frame['files']])

# cargamos las clases de cada imagen
y_train = train_frame['labels_idx'].values
y_test = test_frame['labels_idx'].values
```

Test: Puedes probar si lo has hecho correctamente con el siguiente test, cuando hayas terminado ejecuta la siguiente celda y te debería dar como resultado:

```
 (array([-0.5529412, -0.5529412, -0.5372549], dtype=float32), 3)
```

```
x_train[0,0,0], y_train[0]
```

```
        [-0.14509803, -0.1607843 , -0.15294117],
        [-0.12941176, -0.12941176, -0.12941176],
        [-0.18431371, -0.18431371, -0.18431371],
        [-0.15294117, -0.15294117, -0.15294117],
        [-0.1372549 , -0.1372549 , -0.1372549 ],
        [-0.1372549 , -0.1372549 , -0.15294117],
        [-0.09019607, -0.09019607, -0.10588235],
        [-0.14509803, -0.14509803, -0.1607843 ],
        [-0.12156862, -0.12156862, -0.10588235],
        [-0.16862744, -0.16862744, -0.15294117],
        [-0.15294117, -0.15294117, -0.1372549 ],
        [-0.10588235, -0.12156862, -0.08235294],
        [-0.15294117, -0.15294117, -0.15294117],
        [-0.12941176, -0.12941176, -0.11372548],
        [-0.12156862, -0.1372549 , -0.09803921],
        [-0.12156862, -0.15294117, -0.14509803],
        [-0.12156862, -0.15294117, -0.1607843 ],
        [-0.14509803, -0.17647058, -0.16862744],
        [-0.16862744, -0.19999999, -0.19215685],
        [-0.14509803, -0.1607843 , -0.15294117],
        [-0.11372548, -0.14509803, -0.1372549 ],
        [-0.12941176, -0.1607843 , -0.15294117],
        [-0.1607843 , -0.16862744, -0.18431371],
        [-0.17647058, -0.18431371, -0.19999999],
        [-0.20784312, -0.21568626, -0.23137254],
        [-0.1372549 , -0.14509803, -0.1607843 ],
        [-0.17647058, -0.18431371, -0.19999999],
        [-0.18431371, -0.19215685, -0.15294117],
        [-0.19215685, -0.19215685, -0.19215685],
        [-0.17647058, -0.19215685, -0.18431371],
        [-0.20784312, -0.2235294 , -0.21568626],
        [-0.17647058, -0.19215685, -0.18431371],
        [-0.20784312, -0.2235294 , -0.21568626],
        [-0.15294117, -0.1607843 , -0.17647058],
        [-0.19999999, -0.20784312, -0.23921567],
        [-0.20784312, -0.23921567, -0.26274508],
        [-0.17647058, -0.18431371, -0.19999999],
        [-0.2235294 , -0.23137254, -0.24705881],
        [-0.23137254, -0.24705881, -0.23921567],
        [-0.20784312, -0.20784312, -0.20784312],
        [-0.19999999, -0.19999999, -0.19999999],
        [-0.19215685, -0.19215685, -0.19215685],
        [-0.18431371, -0.18431371, -0.16862744],
        [-0.19999999, -0.19999999, -0.18431371],
        [-0.23137254, -0.23137254, -0.23137254],
        [-0.23921567, -0.23137254, -0.27058822],
        [-0.21568626, -0.2235294 , -0.18431371],
        [-0.23921567, -0.23921567, -0.2235294 ],
        [-0.20784312, -0.20784312, -0.2235294 ],
        [-0.23921567, -0.24705881, -0.26274508],
        [-0.2235294 , -0.23137254, -0.24705881],
        [-0.23137254, -0.23137254, -0.24705881],
        [-0.20784312, -0.21568626, -0.23137254]], dtype=float32),
 3)
```

Ejercicio 3: cargar el modelo pre-entrenado InceptionV3

Una vez tenemos los datos listos, vamos a cargar el modelo pre-entrenado de *InceptionV3*.

Como vamos a aplicar *fine-tuning* recuerda usar los siguientes parámetros:

- `input_shape=(299, 299, 3)`
- `include_top=False`
- `pooling='avg'`

Además tienes que congelar todas las capas para que no se entrenen todas, recureda que solo queremos entrenar las últimas que añadamos nosotros.

```
from tensorflow.keras.applications import InceptionV3

# Cargamos el modelo preentrenado InceptionV3
base_model = InceptionV3(
    input_shape=(299, 299, 3),
    include_top=False,
    pooling='avg'
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering
87910968/87910968 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

+ Código + Texto

Test: Puedes probar si lo has hecho correctamente con el siguiente test, cuando hayas terminado ejecuta la siguiente celda y te debería dar como resultado:

```
 [<keras.layers.merge.Concatenate at **************>,
  <keras.layers.pooling.GlobalAveragePooling2D at **************>]
```

```
base_model.layers[-2:]
```

```
[<Concatenate name=mixed10, built=True>,
 <GlobalAveragePooling2D name=global_average_pooling2d, built=True>]
```

Ejercicio 4: añadir capas al modelo (*fine-tuning*)

Una vez tenemos nuestro modelo base, vamos a añadir capas densas al final para entrenarlas y que el modelo se ajuste a nuestros datos.

Añade las siguientes capas al modelo base cargado:

- Capa Dropout con un valor de 0.60.
- Capa Densa de 128 neuronas y función de activación `relu`.
- Capa Dropout con un valor de 0.4.
- Capa Densa de salida con 5 neuronas y función de activación `softmax`.

```
# Completar
from tensorflow.keras import layers, models


# Crearemos el modelo final añadiendo las capas densas
model = models.Sequential([
    base_model,  # Modelo base congelado
    layers.Dropout(0.6),  # Capa Dropout con un valor de 0.6
    layers.Dense(128, activation='relu'),  # Capa densa con 128 neuronas y activación ReLU
    layers.Dropout(0.4),  # Capa Dropout con un valor de 0.4
    layers.Dense(5, activation='softmax')  # Capa de salida con 5 neuronas y activación Softmax
])
```

Test: Puedes probar si lo has hecho correctamente con el siguiente test, cuando hayas terminado ejecuta la siguiente celda y te debería dar como resultado:

```
 [<keras.layers.core.dropout.Dropout at **************>,
  <keras.layers.core.dense.Dense at **************>,
  <keras.layers.core.dropout.Dropout at **************>,
  <keras.layers.core.dense.Dense at **************>]
```

```
model.layers[-4:]
```

```
[<Dropout name=dropout, built=True>,
 <Dense name=dense, built=True>,
 <Dropout name=dropout_1, built=True>,
 <Dense name=dense_1, built=True>]
```

Ejercicio 5: entrenar el modelo

Una vez tenemos nuestro modelo listo para entrenar, vamos a configurar el entrenamiento y a entrenar nuesto modelo.

En el entrenamiento utiliza:

- Optimizador: Adam con learning rate de 0.001.
- Función de coste: `sparse_categorical_crossentropy`.
- Métricas: `accuracy`.
- Epochs: `25`
- validation_split: 0.2

```
# Completar

import numpy as np

# Eliminaremo la dimensión adicional en x_train si existe
x_train = np.squeeze(x_train)  # Remueve cualquier dimensión de tamaño 1

# Asegúrarse de que x_train ahora tenga forma (num_samples, 299, 299, 3)
print("Forma de x_train:", x_train.shape)
```

```
# Compilamos el modelo
from tensorflow.keras.optimizers import Adam

model.compile(
    optimizer=Adam(learning_rate=0.001),  # Optimizador Adam
    loss='sparse_categorical_crossentropy',  # Función de coste
    metrics=['accuracy']  # Métrica de evaluación
)

# Entrenamos el modelo
hist = model.fit(
    x_train,  # Conjunto de entrenamiento con forma correcta
    y_train,  # Etiquetas del conjunto de entrenamiento
    epochs=25,  # Número de épocas
    validation_split=0.2,  # Porcentaje del conjunto para validación
    batch_size=32,  # Tamaño del batch
    verbose=1  # Mostrar progreso
)
```

```
Forma de x_train: (1385, 299, 299, 3)
Epoch 1/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 188s 3s/step - accuracy: 0.3931 - loss: 1.6324 - val_accuracy: 0.1841 - val_loss: 59.8479
Epoch 2/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 63s 451ms/step - accuracy: 0.5187 - loss: 1.2975 - val_accuracy: 0.1841 - val_loss: 92.9224
Epoch 3/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 20s 443ms/step - accuracy: 0.5638 - loss: 1.1954 - val_accuracy: 0.2238 - val_loss: 9.3135
Epoch 4/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 15s 438ms/step - accuracy: 0.6426 - loss: 1.0067 - val_accuracy: 0.2058 - val_loss: 4607.6416
Epoch 5/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 15s 434ms/step - accuracy: 0.6677 - loss: 0.9093 - val_accuracy: 0.4838 - val_loss: 6.1856
Epoch 6/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 15s 440ms/step - accuracy: 0.6775 - loss: 0.8580 - val_accuracy: 0.3899 - val_loss: 2.4420
Epoch 7/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 21s 443ms/step - accuracy: 0.7128 - loss: 0.7581 - val_accuracy: 0.3827 - val_loss: 3.9387
Epoch 8/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 21s 449ms/step - accuracy: 0.7411 - loss: 0.7240 - val_accuracy: 0.4152 - val_loss: 5.4955
Epoch 9/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 21s 458ms/step - accuracy: 0.7789 - loss: 0.6336 - val_accuracy: 0.4043 - val_loss: 2.4777
Epoch 10/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 21s 459ms/step - accuracy: 0.7592 - loss: 0.6392 - val_accuracy: 0.4404 - val_loss: 6.8854
Epoch 11/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 16s 454ms/step - accuracy: 0.8476 - loss: 0.4256 - val_accuracy: 0.5812 - val_loss: 2.8923
Epoch 12/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 16s 446ms/step - accuracy: 0.8297 - loss: 0.4972 - val_accuracy: 0.3899 - val_loss: 5.0431
Epoch 13/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 16s 445ms/step - accuracy: 0.8483 - loss: 0.4371 - val_accuracy: 0.5560 - val_loss: 1.6308
Epoch 14/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 16s 443ms/step - accuracy: 0.9038 - loss: 0.2887 - val_accuracy: 0.4332 - val_loss: 2.8178
Epoch 15/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 20s 433ms/step - accuracy: 0.8885 - loss: 0.3629 - val_accuracy: 0.4296 - val_loss: 4.0279
Epoch 16/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 21s 439ms/step - accuracy: 0.9423 - loss: 0.2024 - val_accuracy: 0.3718 - val_loss: 6.6081
Epoch 17/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 20s 439ms/step - accuracy: 0.8200 - loss: 0.4914 - val_accuracy: 0.4368 - val_loss: 3.3662
Epoch 18/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 15s 440ms/step - accuracy: 0.9187 - loss: 0.2783 - val_accuracy: 0.4440 - val_loss: 5.0584
Epoch 19/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 15s 437ms/step - accuracy: 0.9461 - loss: 0.1448 - val_accuracy: 0.5271 - val_loss: 3.4963
Epoch 20/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 21s 440ms/step - accuracy: 0.9539 - loss: 0.1485 - val_accuracy: 0.5018 - val_loss: 3.8757
Epoch 21/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 20s 436ms/step - accuracy: 0.9521 - loss: 0.2108 - val_accuracy: 0.5307 - val_loss: 4.4859
Epoch 22/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 21s 448ms/step - accuracy: 0.9321 - loss: 0.2340 - val_accuracy: 0.5596 - val_loss: 1.9121
Epoch 23/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 20s 438ms/step - accuracy: 0.9367 - loss: 0.1851 - val_accuracy: 0.5812 - val_loss: 2.6997
Epoch 24/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 20s 436ms/step - accuracy: 0.9654 - loss: 0.1228 - val_accuracy: 0.5560 - val_loss: 3.2026
Epoch 25/25
35/35 ━━━━━━━━━━━━━━━━━━━━ 21s 444ms/step - accuracy: 0.9573 - loss: 0.1164 - val_accuracy: 0.4946 - val_loss: 3.9079
```

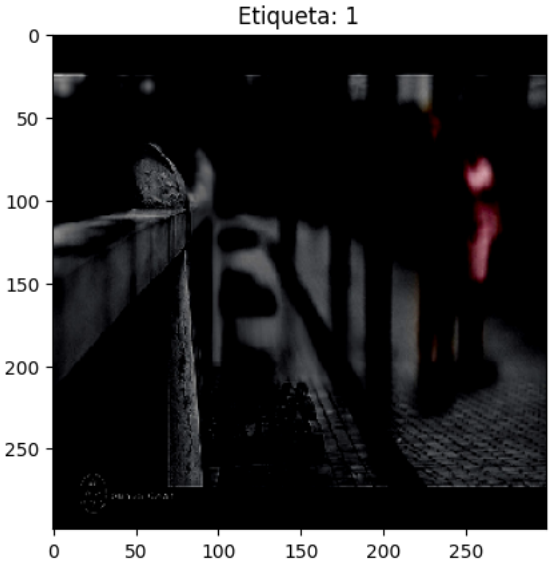
Ejercicio 6: evaluar el modelo

Una vez entrenado el modelo usando *fine-tuning* evalua el modelo usando el conjunto de test en la función `evaluate` y extráe conlcusiones de si el modelo tiene un buen rendimiento o no. Puedes visualizar como ha ido el entrenamiento usando una gráfica como hemos visto en clase.

```
# Completar

print(f"Forma de x_test: {x_test.shape}")  # Debería ser (n_samples, 299, 299, 3)
print(f"Forma de y_test: {y_test.shape}")  # Debería ser (n_samples,)

x_test = np.squeeze(x_test)  # Elimina dimensiones de tamaño 1 si es necesario
```

```
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=1)
print(f"Pérdida en el conjunto de prueba: {test_loss}")
print(f"Precisión en el conjunto de prueba: {test_accuracy}")

import matplotlib.pyplot as plt

plt.imshow(x_test[0])  # Muestra la primera imagen
plt.title(f"Etiqueta: {y_test[0]}")
plt.show()
```

```
Forma de x_test: (320, 1, 299, 299, 3)
Forma de y_test: (320,)
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 118ms/step - accuracy: 0.4913 - loss: 4.2731
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers
Pérdida en el conjunto de prueba: 4.607175827026367
Precisión en el conjunto de prueba: 0.484375
```




```
import matplotlib.pyplot as plt # Importamos para generar grafico

# Gráfico de precisión
plt.plot(hist.history['accuracy'], label='Precision de entrenamiento')
plt.plot(hist.history['val_accuracy'], label='Precisión de validacion')
plt.xlabel('epoch')
plt.ylabel('Precision')
plt.legend()
plt.title('Precisión durante el entrenamiento')
plt.show()
```