

AREA MODELOS DE PYTHON A TENSORFLOW.JS

Néstor Batista Díaz

1 De grados Fahrenheita a Celsius

```
[54]: import tensorflow as tf
import numpy as np
import random
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
```

Generamos 1000 filas con los grados Fahrenheit y sus respectivos grados Celsius

```
[55]: # Crea un DataFrame vacío para almacenar los valores
df = pd.DataFrame(columns=['Celsius', 'Fahrenheit'])

# Genera 1000 valores aleatorios de Celsius
for i in range(1000):
    celsius = random.uniform(-100, 100)

    # Convierte Celsius a Fahrenheit
    fahrenheit = (celsius * 9/5) + 32

    # Agrega los valores al DataFrame
    df.loc[i] = [round(celsius,2), round(fahrenheit,2)]

df.to_csv("Grados/grados.csv")
```

Creamos el modelo según el video y lo entrenamos

```
[56]: df = pd.read_csv("Grados/grados.csv")

y = df["Celsius"]
X = df["Fahrenheit"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
↪random_state=42)

oculta1 = tf.keras.layers.Dense(units=3, input_shape=[1])
```

```

oculta2 = tf.keras.layers.Dense(units=3)
salida = tf.keras.layers.Dense(units=1)
modelo = tf.keras.Sequential([oculta1, oculta2, salida])

modelo.compile(
    optimizer=tf.keras.optimizers.Adam(0.01),
    loss='mean_squared_error'
)

print("Comenzando entrenamiento...")
historial = modelo.fit(X_train, y_train, epochs=300, verbose=False,
    ↪validation_split=0.05)
print("Modelo entrenado!")

```

```

[56]: Comenzando entrenamiento...
      Modelo entrenado!

```

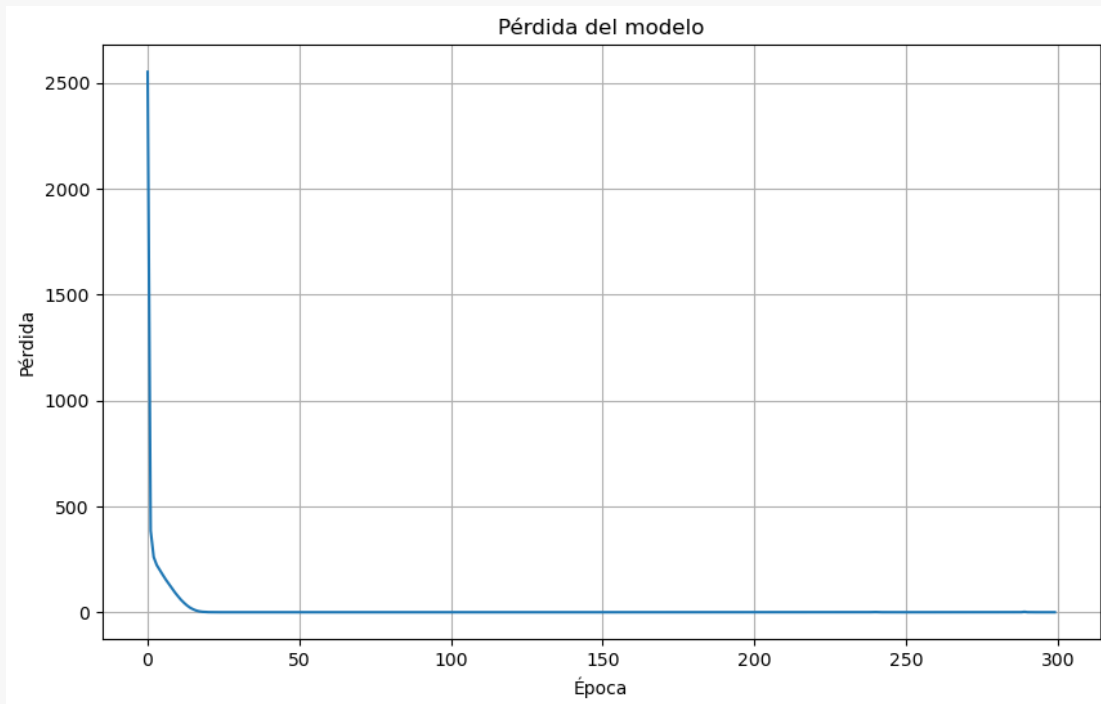
```

[57]: # Obtener la historia de entrenamiento del modelo
      loss = historial.history['loss']

      # Crear la gráfica de la pérdida
      plt.figure(figsize=(10, 6))
      plt.plot(loss)
      plt.title('Pérdida del modelo')
      plt.xlabel('Época')
      plt.ylabel('Pérdida')
      plt.grid(True)
      plt.show()

```

[57]:



Como podemos ver el modelo aprende muy rapido

```
[58]: mse = modelo.evaluate(X_test, y_test, batch_size=32, verbose=0)

print("MSE: ", mse)
```

```
[58]: MSE:  1.3885900443710852e-05
```

El error del modelo es muy bajo

```
[59]: resultado = modelo.predict(np.array([100]), verbose=0)
print("Predicción: 100 grados Fahrenheit son {:.2f} Celsius!".
      ↪format(resultado[0][0]))
```

```
[59]: Predicción: 100 grados Fahrenheit son 37.78 Celsius!
```

Guardamos el modelo para pasarselo a tensorflowjs y nos genere el json del modelo

```
[60]: #Exportar el modelo en formato h5
modelo.save('Grados/fahrenheit_a_celsius.h5')
```

```
[60]: c:\Users\NestorBD\anaconda3\Lib\site-packages\keras\src\engine\training.py:3103:
UserWarning: You are saving your model as an HDF5 file via `model.save()`. This
file format is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
```

```
saving_api.save_model(
```

El siguiente paso lo ejecute en Google Colab ya que en local no funciona el tensorflowjs por un problema de versiones.

[Google Colab](#)

Subimos el modelo guardado anteriormente

```
[1] from google.colab import files

uploaded = files.upload()

Elegir archivos fahrenheit_a_celsius.h5
• fahrenheit_a_celsius.h5(n/a) - 32192 bytes, last modified: 24/4/2024 - 100% done
Saving fahrenheit_a_celsius.h5 to fahrenheit_a_celsius.h5
```

Creamos la carpeta de salida e instalamos tensorflowjs

```
#Crear carpeta donde se colocaran los archivos resultantes
mkdir output

[3] #Para convertirlo a tensorflow.js, primero debemos instalar la libreria
!pip install tensorflowjs

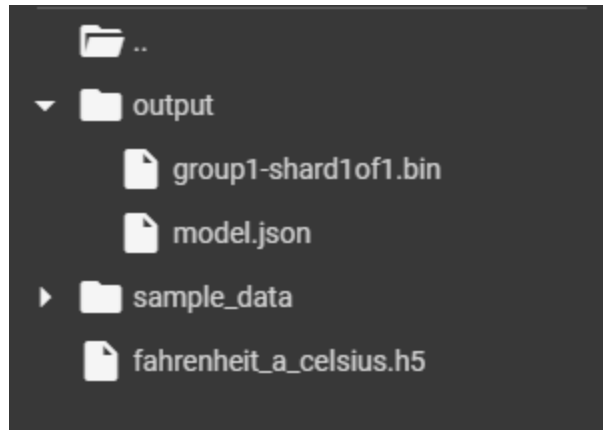
Collecting tensorflowjs
  Downloading tensorflowjs-4.18.0-py3-none-any.whl (89 kB)
    89.1/89.1 kB 1.9 MB/s eta 0:00:00
Requirement already satisfied: flax>=0.7.2 in /usr/local/lib/python3.10/dist-packages (from tensorflowjs) (0.8.2)
Requirement already satisfied: importlib_resources>=5.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflowjs) (6.4.0)
Requirement already satisfied: jax>=0.4.13 in /usr/local/lib/python3.10/dist-packages (from tensorflowjs) (0.4.26)
Requirement already satisfied: jaxlib>=0.4.13 in /usr/local/lib/python3.10/dist-packages (from tensorflowjs) (0.4.26+cuda12.cudnn89)
Requirement already satisfied: tensorflow>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from tensorflowjs) (2.15.0)
Requirement already satisfied: tf-keras>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from tensorflowjs) (2.15.1)
Collecting tensorflow-decision-forests>=1.5.0 (from tensorflowjs)
  Downloading tensorflow-decision-forests-1.9.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (15.5 MB)
    15.5/15.5 MB 27.2 MB/s eta 0:00:00
Requirement already satisfied: six<2,>=1.16.0 in /usr/local/lib/python3.10/dist-packages (from tensorflowjs) (1.16.0)
Requirement already satisfied: tensorflow-hub>=0.16.1 in /usr/local/lib/python3.10/dist-packages (from tensorflowjs) (0.16.1)
Collecting packaging~>23.1 (from tensorflowjs)
  Downloading packaging-23.2-py3-none-any.whl (53 kB)
    53.0/53.0 kB 4.5 MB/s eta 0:00:00
```

Con tensorflowjs generamos los archivos json y bin necesarios para mostrar los resultados en la web

```
#Realizar la exportacion a la carpeta de salida
!tensorflowjs_converter --input_format keras fahrenheit_a_celsius.h5 output

2024-04-24 16:23:06.095258: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use avx2
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2024-04-24 16:23:07.622055: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
```

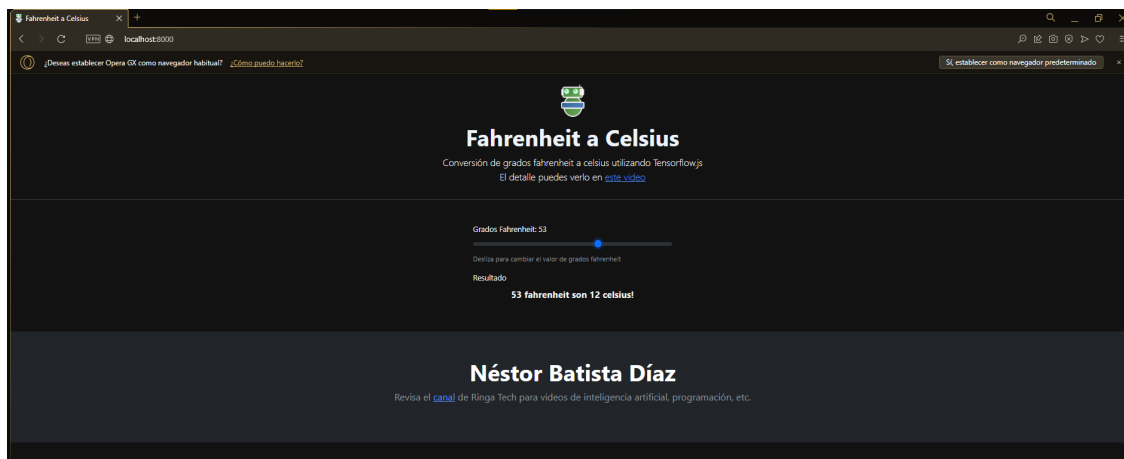
Aqui podemos ver los archivos generados, ahora los descargamos en la carpeta donde tenemos guardado la pagina web



Abrimos el servidor en la carpeta donde tenemos guardado la pagina web

```
$ python -m http.server 8000
Serving HTTP on :: port 8000 (http://[::]:8000/) ...
::1 - - [24/Apr/2024 17:24:20] "GET / HTTP/1.1" 304 -
::1 - - [24/Apr/2024 17:24:20] "GET /model.json HTTP/1.1" 200 -
::1 - - [24/Apr/2024 17:24:20] "GET /group1-shard1of1.bin HTTP/1.1" 200 -
```

Y por ultimo abrimos el siguiente [link](#) en el navegador



2 Clasificación de flores

```
[1]: import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os
import numpy as np
import matplotlib.pyplot as plt
import splitfolders
```

```
import cv2
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report
```

```
[1]: WARNING:tensorflow:From c:\Users\NestorBD\anaconda3\Lib\site-
packages\keras\src\losses.py:2976: The name
tf.losses.sparse_softmax_cross_entropy is deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

Definimos la carpeta en donde se encuentran las imagenes con las que vamos a entrenar el modelo

```
[2]: original_path = "Flores/flowers"
print(os.listdir(original_path))
```

```
[2]: ['daisy', 'dandelion', 'rose', 'sunflower', 'tulip']
```

Creamos un MAP con las especies de flores que tenemos para entrenar y definimos el tamaño de las imagenes

```
[3]: MAP_FLOWERS = {
    0: 'daisy', 1: 'dandelion', 2: 'rose',
    3: 'sunflower', 4: 'tulip'
}

# Vamos a standarizar todas las imágenes a tamaño 128x128
IMG_SIZE = 128
```

Dividimos los archivos en tres conjuntos entrenamiento, validación y pruebas

```
[4]: splitfolders.ratio('Flores/flowers',seed=1337, output="Flores/Flowers-Splitted",
    ↪ratio=(0.6, 0.2, 0.2))
```

```
[4]: Copying files: 2131 files [00:33, 60.96 files/s]
```

Creamos las funciones para cargar las imagenes

```
[6]: def load_train_set(dirname, map_pieces, verbose=True):
    """Esta función carga los datos de training en imágenes.

    Como las imágenes tienen tamaños distintas, utilizamos la librería opencv
    para hacer un resize y adaptarlas todas a tamaño IMG_SIZE x IMG_SIZE.

    Args:
        dirname: directorio completo del que leer los datos
        map_pieces: variable de mapeo entre labels y piezas
        verbose: si es True, muestra información de las imágenes cargadas

    Returns:
```

```

        X, y: X es un array con todas las imágenes cargadas con tamaño
              IMG_SIZE x IMG_SIZE
              y es un array con las labels de correspondientes a cada imagen
    """
    X_train = []
    y_train = []
    for label, piece in map_pieces.items():
        files = os.listdir(os.path.join(dirname, piece))
        images = [file for file in files if (file.endswith("jpg") or file.
↪endswith("png") or file.endswith("jpeg") or file.endswith("JPG"))]
        if verbose:
            print("Leyendo {} imágenes encontradas de {}".format(len(images),
↪piece))
        for image_name in images:
            image = cv2.imread(os.path.join(dirname, piece, image_name))
            X_train.append(cv2.resize(image, (IMG_SIZE, IMG_SIZE)))
            y_train.append(label)
    return np.array(X_train), np.array(y_train)

```

```

[7]: def load_test_or_val_set(dirname, map_pieces, verbose=True, isTest=True):
        """Esta función funciona de manera equivalente a la función load_train_set
        pero cargando los datos de test."""
        X_test = []
        y_test = []
        for label, piece in map_pieces.items():
            files = os.listdir(os.path.join(dirname, piece))
            images = [file for file in files if (file.endswith("jpg") or file.
↪endswith("png") or file.endswith("jpeg") or file.endswith("JPG"))]
            for image_name in images:
                image = cv2.imread(os.path.join(dirname, piece, image_name))
                X_test.append(cv2.resize(image, (IMG_SIZE, IMG_SIZE)))
                y_test.append(label)
        if verbose:
            if isTest:
                print("Leídas {} imágenes de test".format(len(X_test)))
            else:
                print("Leídas {} imágenes de val".format(len(X_test)))
        return np.array(X_test), np.array(y_test)

```

Cargamos los datos.

```

[10]: DATASET_TRAIN_PATH = "Flores/Flowers-Splitted/train"
        DATASET_VAL_PATH = "Flores/Flowers-Splitted/val"
        DATASET_TEST_PATH = "Flores/Flowers-Splitted/test"

        X, y = load_train_set(DATASET_TRAIN_PATH, MAP_FLOWERS)
        X_v, y_v = load_test_or_val_set(DATASET_TEST_PATH, MAP_FLOWERS, isTest=False)

```

```
X_t, y_t = load_test_or_val_set(DATASET_TEST_PATH, MAP_FLOWERS)
```

```
[10]: Leyendo 300 imágenes encontradas de daisy
Leyendo 387 imágenes encontradas de dandelion
Leyendo 298 imágenes encontradas de rose
Leyendo 297 imágenes encontradas de sunflower
Leyendo 364 imágenes encontradas de tulip
Leídas 552 imágenes de val
Leídas 552 imágenes de test
```

Aumentamos los datos de entrenamiento

```
[11]: #Aumento de datos
image_gen_entrenamiento = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')
```

```
[12]: #Generacion de datos de entrenamiento FTW
data_gen_entrenamiento = image_gen_entrenamiento.
    ↳flow_from_directory(batch_size=100,
                                directory="Flores/
    ↳Flowers-Splitted/train",
                                shuffle=True,
    ↳target_size=(IMG_SIZE,IMG_SIZE),
                                class_mode='categorical')
```

```
[12]: Found 1646 images belonging to 5 classes.
```

Rescalamos los datos de validación para que sean iguales a los de entrenamiento.

```
[13]: #Generacion de datos de validacion
image_gen_val = ImageDataGenerator(rescale=1./255)

data_gen_validacion = image_gen_val.flow_from_directory(batch_size=100,
                                                        directory="Flores/
    ↳Flowers-Splitted/val",
                                                        target_size=(IMG_SIZE,
    ↳IMG_SIZE),
                                                        class_mode='categorical')
```

```
[13]: Found 548 images belonging to 5 classes.
```


Craemos el modelo muy sencillo, tal cual esta en el video, solo cambiando los datos de salida, sino puede dar fallos.

```
[42]: modelo = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(IMG_SIZE,
↪IMG_SIZE, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(5, activation='softmax')
])
```

Compilación y entrenamiento

```
[43]: #Compilación
modelo.compile(optimizer='adam',
               loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
               metrics=['accuracy'])

#Entrenar la red. Toma un buen rato! Ve por un café ;)
#Oye suscribete al canal!
print("Entrenando modelo...");
epocas=60
history = modelo.fit(
    data_gen_entrenamiento,
    steps_per_epoch=int(np.ceil(len(X) / float(100))),
    epochs=epocas,
    validation_data=data_gen_validacion,
    validation_steps=int(np.ceil(len(X_v) / float(100)))
)

print("Modelo entrenado!");
```

```
[43]: Entrenando modelo...
Epoch 1/60
c:\Users\NestorBD\anaconda3\Lib\site-packages\keras\src\backend.py:5575:
UserWarning: "`categorical_crossentropy` received `from_logits=True`, but the
```

```

`output` argument was produced by a Softmax activation and thus does not
represent logits. Was this intended?
    output, from_logits = _get_logits(
17/17 ----- 37s 2s/step - loss: 1.5333 - accuracy:
0.2807 - val_loss: 1.2777 - val_accuracy: 0.4507
Epoch 2/60
17/17 ----- 33s 2s/step - loss: 1.2364 - accuracy:
0.4459 - val_loss: 1.2061 - val_accuracy: 0.4653
Epoch 3/60
17/17 ----- 33s 2s/step - loss: 1.1604 - accuracy:
0.5200 - val_loss: 1.1896 - val_accuracy: 0.5182
Epoch 4/60
17/17 ----- 33s 2s/step - loss: 1.0834 - accuracy:
0.5462 - val_loss: 1.0571 - val_accuracy: 0.5803
Epoch 5/60
17/17 ----- 34s 2s/step - loss: 1.1077 - accuracy:
0.5504 - val_loss: 1.0573 - val_accuracy: 0.5584
.....
Epoch 55/60
17/17 ----- 41s 2s/step - loss: 0.4568 - accuracy:
0.8311 - val_loss: 0.6196 - val_accuracy: 0.7810
Epoch 56/60
17/17 ----- 37s 2s/step - loss: 0.4396 - accuracy:
0.8262 - val_loss: 0.6075 - val_accuracy: 0.7938
Epoch 57/60
17/17 ----- 36s 2s/step - loss: 0.4176 - accuracy:
0.8469 - val_loss: 0.6310 - val_accuracy: 0.7701
Epoch 58/60
17/17 ----- 38s 2s/step - loss: 0.4362 - accuracy:
0.8275 - val_loss: 0.7968 - val_accuracy: 0.7810
Epoch 59/60
17/17 ----- 35s 2s/step - loss: 0.4145 - accuracy:
0.8451 - val_loss: 0.6386 - val_accuracy: 0.7719
Epoch 60/60
17/17 ----- 40s 2s/step - loss: 0.4002 - accuracy:
0.8481 - val_loss: 0.6365 - val_accuracy: 0.7847
Modelo entrenado!

```

```

[44]: def plot_acc(history, title="Model Accuracy"):
    """Imprime una gráfica mostrando la accuracy por epoch obtenida en un
    ↪entrenamiento"""
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title(title)
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='upper left')

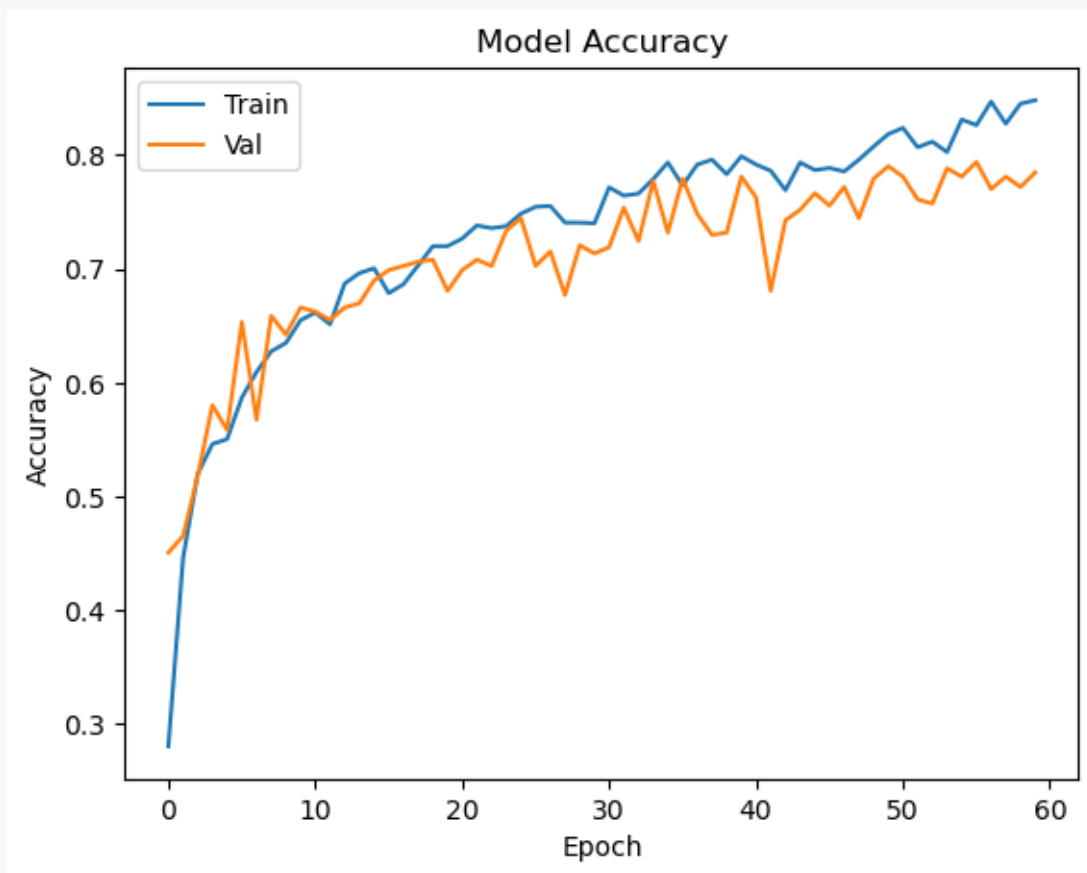
```

```
plt.show()
def plot_loss(history, title="Model Loss"):
    """Imprime una gráfica mostrando la pérdida por epoch obtenida en un
→entrenamiento"""
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title(title)
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='upper right')
    plt.show()
```

En la siguiente grafica podemos observar la variacion de aciertos de los datos de entrenamiento con respecto a los de validación, van mas o menos a la par.

```
[45]: plot_acc(history)
```

```
[45]:
```

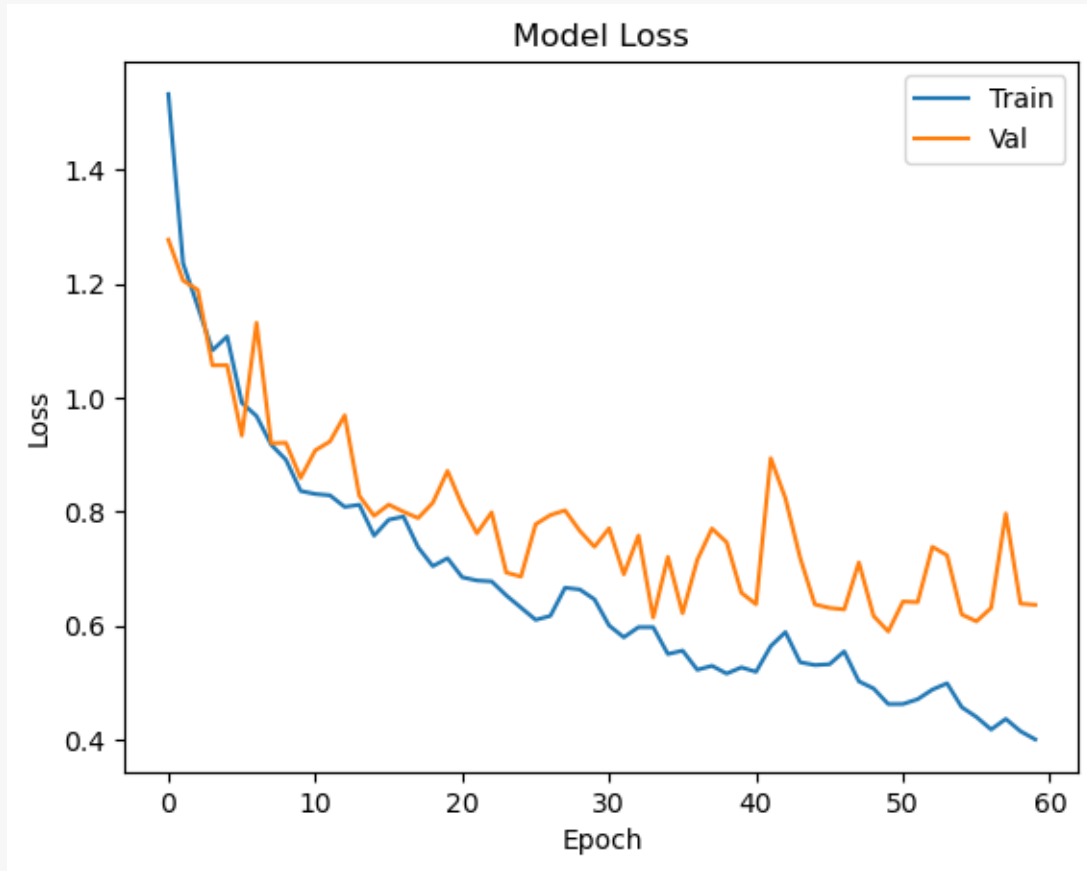


En este otro grafico vemos la misma comparación pero con los datos de pérdida, vemos que a partir de la epoca 50 se empiezan a separar mucho, se recomendaria parar ahi o incluso en la 40 donde

ya se aprecia una separación considerable.

```
[46]: plot_loss(history)
```

[46]:



Normalizamos los datos de test para comprobar nuestro modelo entrenado.

```
[47]: #Generacion de datos de test
image_gen_test = ImageDataGenerator(rescale=1./255)

data_gen_test = image_gen_test.flow_from_directory(batch_size=100,
                                                    directory="Flores/
↳Flowers-Splitted/test",
                                                    target_size=(IMG_SIZE,
↳IMG_SIZE),
                                                    class_mode='categorical')

# Obtener un lote de datos
X_test, y_test = data_gen_test.next()
```

[47]: Found 552 images belonging to 5 classes.

Evaluamos el modelo y comprobamos que sale correctamente, el resultado es bastante aceptable, lo que se esperaba.

```
[48]: modelo.evaluate(X_test, y_test, batch_size=32, verbose=1)
```

```
[48]: 4/4 ----- - 0s 35ms/step - loss: 0.7134 - accuracy:
0.7600
[0.713397741317749, 0.7599999904632568]
```

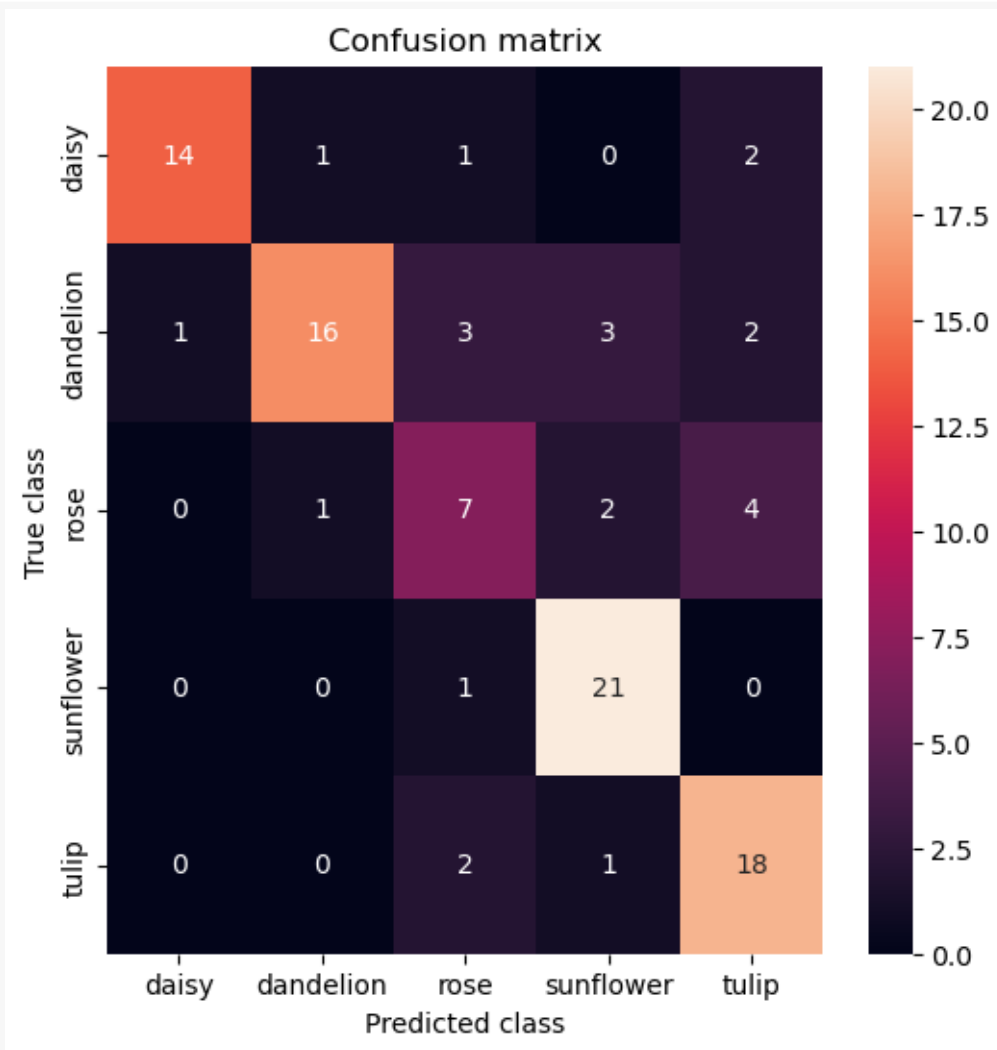
Generamos la matriz de confusión

```
[49]: LABELS = ['daisy', 'dandelion', 'rose',
'sunflower', 'tulip']
def mostrar_resultados(model, X_test, y_test, pred_y):
    pred_y = np.argmax(pred_y, axis=1)
    y_test = np.argmax(y_test, axis=1)
    conf_matrix = confusion_matrix(y_test, pred_y)
    plt.figure(figsize=(6, 6))
    sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True,
↪fmt="d");
    plt.title("Confusion matrix")
    plt.ylabel('True class')
    plt.xlabel('Predicted class')
    plt.show()

    print(classification_report(y_test, pred_y))
```

```
[50]: y_pred = modelo.predict(X_test)
mostrar_resultados(history, X_test, y_test, y_pred)
```

```
[50]: 4/4 ----- - 0s 33ms/step
```



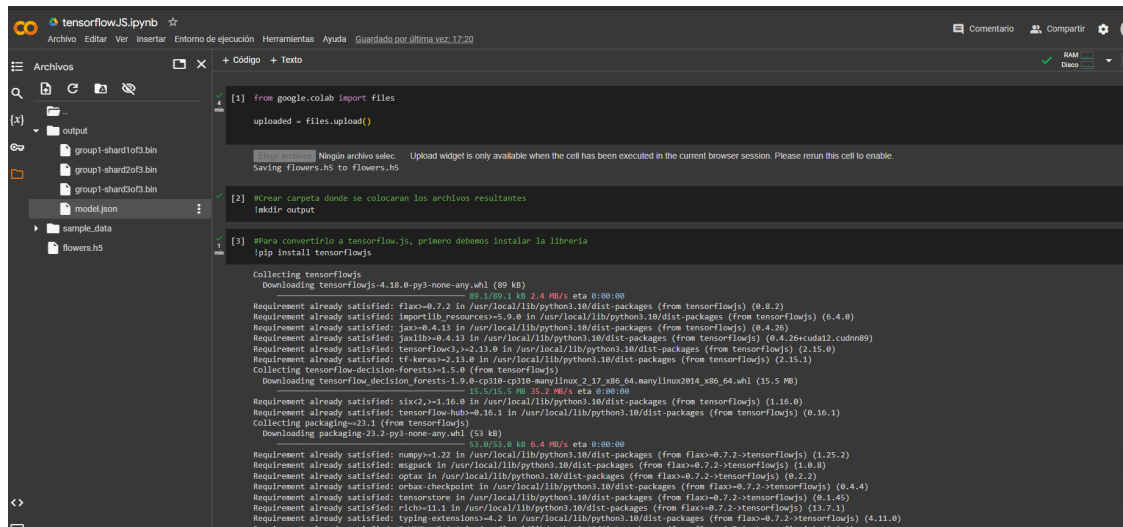
	precision	recall	f1-score	support
0	0.93	0.78	0.85	18
1	0.89	0.64	0.74	25
2	0.50	0.50	0.50	14
3	0.78	0.95	0.86	22
4	0.69	0.86	0.77	21
accuracy			0.76	100
macro avg	0.76	0.75	0.74	100
weighted avg	0.78	0.76	0.76	100

En la matriz podemos comprobar que acierta bastante en todas las categorías a excepción de la rosa, lo cual es aceptable para esta tarea de clase

Exportamos el modelo

```
[53]: #Exportar el modelo en formato h5
modelo.save('Flores/flowers.h5')
```

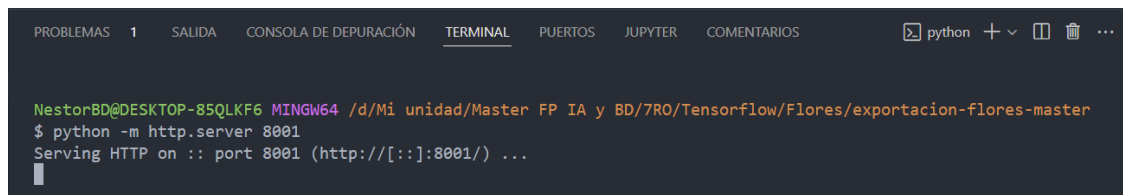
Ahora, seguimos los mismos pasos que con el modelo anterior, subimos el modelo a el codigo de google colab para generar los archivos necesarios.



The screenshot shows the Google Colab interface. On the left, the 'Archivos' (Files) pane shows a directory structure with 'output' containing 'group1-shard1of3.bin', 'group1-shard2of3.bin', 'group1-shard3of3.bin', 'model.json', 'sample_data', and 'flowers.h5'. The main code area contains three cells: Cell 1 imports files from Google Colab and uploads them; Cell 2 creates an output directory; Cell 3 installs TensorFlow.js using 'pip install tensorflowjs'. The output of Cell 3 shows the installation progress and requirements for TensorFlow.js, including downloading various wheels and packages like 'tensorflowjs-4.18.0-py3-none-any.whl', 'flax-0.7.2', 'jax-0.4.15', 'jaxlib-0.4.13', 'tensorflow-cpu-2.13.0', 'tensorflow-decision-forests-1.5.0', 'tensorflow-hub-0.16.1', 'packaging-21.2', 'numpy-1.22.1', 'msgpack-1.0', 'optax-0.2.2', 'orbax-checkpoint-0.4.4', 'tensorstore-0.1.45', 'rich-11.1', 'typing-extensions-4.2', and 'pydantic-1.9.0'.

Descargamos los archivos de la carpeta output y los añadimos a la carpeta en donde se encuentra el html donde mostraremos el funcionamiento del modelo con la camara del movil conectado al ordenador.

Abrimos el servidor en la carpeta donde tenemos guardado la pagina web



The screenshot shows a terminal window with the following output: 'NestorBD@DESKTOP-85QLKF6 MINGW64 /d/Mi unidad/Master FP IA y BD/7RO/Tensorflow/Flores/exportacion-flores-master', '\$ python -m http.server 8001', and 'Serving HTTP on :: port 8001 (http://[::]:8001/) ...'.

Y por ultimo abrimos el [link](#) en donde hemos alojado la web

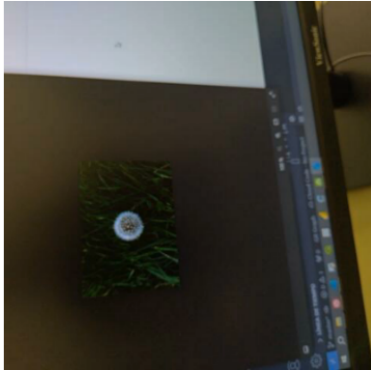
localhost:8001

Buscar en Bing

Flores

Clasificación de imágenes usando la cámara web utilizando
Tensorflow.js
El detalle puedes verlo en [este video](#)

Cambiar cámara



dandelion

Néstor Batista Díaz

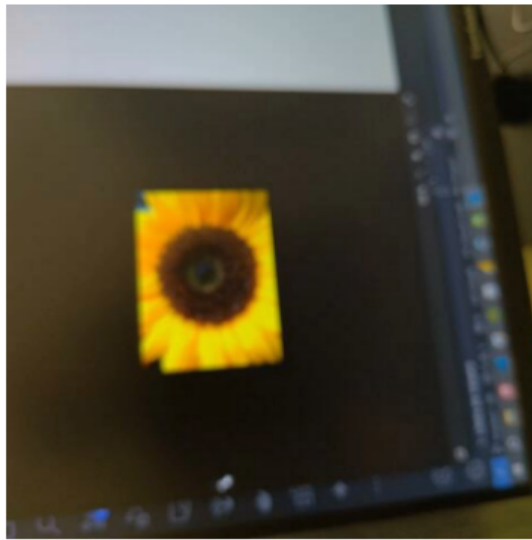
Flores

Clasificación de imágenes usando la cámara web utilizando

Tensorflow.js

El detalle puedes verlo en [este video](#)

Cambiar cámara



sunflower

Néstor Batista Díaz