Tensorflow.JS

[Subtítulo del documento]

Javier Díaz Machado

1 (4 Puntos) Realiza la tarea de implementar un modelo para convertir temperaturas de grados Fahrenheit a centígrados. Expórtalo a Tensorflow.js e implementa la aplicación web para que use el modelo.

Compilamos y entrenamos el modelo:

```
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...")
history = modelo.fit(celsius_train_final, fahrenheit_train_final,epochs=300,
                     verbose=False,validation_data=(celsius_val, fahrenheit_val))
print("Modelo entrenado!")
```

Descarga la función de conversión y genera el dataset .csv con al menos 1000 temperaturas.

```
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',
    metrics=[regression_accuracy]
)

print("Comenzando entrenamiento...")
history = modelo.fit(celsius_train_final, fahrenheit_train_final,epochs=300,
                     verbose=False,validation_data=(celsius_val, fahrenheit_val))
print("Modelo entrenado!")
```

Divide los datos en 80% training y 20% test. Los datos de trainning reserva un 5% para validación.

```
# Tus datos originales
celsius = np.array([-40, -10, 0, 8, 15, 22, 38], dtype=float)
fahrenheit = np.array([-40, 14, 32, 46, 59, 72, 100], dtype=float)

# 1. Divide en 80% train y 20% test
celsius_train, celsius_test, fahrenheit_train, fahrenheit_test = train_test_split(
    celsius, fahrenheit, test_size=0.2, random_state=42
)

# 2. Reserva 5% del train para validación
celsius_train_final, celsius_val, fahrenheit_train_final, fahrenheit_val = train_test_split(
    celsius_train, fahrenheit_train, test_size=0.05, random_state=42
)
```

Muestra las gráficas de pérdida y precisión.

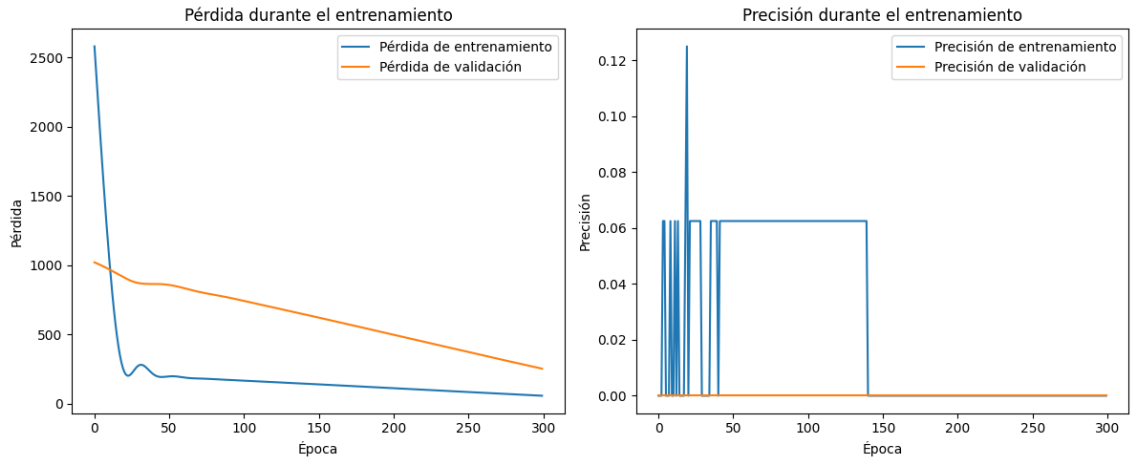
```python
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 5))

# Gráfica de pérdida (loss)
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Pérdida de entrenamiento')
plt.plot(history.history['val_loss'], label='Pérdida de validación')
plt.xlabel('Época')
plt.ylabel('Pérdida')
plt.title('Pérdida durante el entrenamiento')
plt.legend()

# Gráfica de precisión (accuracy)
plt.subplot(1, 2, 2)
plt.plot(history.history['regression_accuracy'], label='Precisión de entrenamiento')
plt.plot(history.history['val_regression_accuracy'], label='Precisión de validación')
plt.xlabel('Época')
plt.ylabel('Precisión')
plt.title('Precisión durante el entrenamiento')
plt.legend()

plt.tight_layout()
plt.show()
```



2 (6 Puntos) Descarga el data set de flores de:

https://www.kaggle.com/datasets/imsparsh/flowers-dataset?resource=download

```python
import kagglehub
import os
import shutil
# Download latest version
path = kagglehub.dataset_download("imsparsh/flowers-dataset")

print("Path to dataset files:", path)
# Copiar recursivamente todas las carpetas y archivos de 'path' a 'dataset' sin función recursiva
if not os.path.exists("dataset"):
    os.makedirs("dataset")
for root, dirs, files in os.walk(path):
    # Construir la ruta de destino correspondiente
    dest_dir = os.path.join("dataset", os.path.relpath(root, path))
    if not os.path.exists(dest_dir):
        os.makedirs(dest_dir)
    for file in files:
        src_file = os.path.join(root, file)
        dst_file = os.path.join(dest_dir, file)
        shutil.copy2(src_file, dst_file)
```

Implementa en Python el modelo de red convolucional que clasifique correctamente las flores, expórtalo y úsalo en una aplicación web en la que se seleccionará una imagen e indicará su nombre.

```
import tensorflow as tf

IMG_WIDTH = 180
IMG_HEIGHT = 180
BATCH_SIZE = 32

# Cargar datasets desde la carpeta 'dataset/'
train_ds = tf.keras.utils.image_dataset_from_directory(
   "dataset",
   validation_split=0.2,
   subset="training",
   seed=123,
   image_size=(IMG_HEIGHT, IMG_WIDTH),
   batch_size=BATCH_SIZE
)
val_ds = tf.keras.utils.image_dataset_from_directory(
   "dataset",
   validation_split=0.2,
   subset="validation",
   seed=123,
   image_size=(IMG_HEIGHT, IMG_WIDTH),
   batch_size=BATCH_SIZE
)

# Obtener nombres de las clases
class_names = train_ds.class_names
print(class_names)
```

```
import tensorflow as tf
import json

IMG_WIDTH = 180
IMG_HEIGHT = 180
BATCH_SIZE = 32

# Cargar datasets desde la carpeta 'dataset/train/'
train_ds = tf.keras.utils.image_dataset_from_directory(
   "dataset/train",
   validation_split=0.2,
   subset="training",
   seed=123,
   image_size=(IMG_HEIGHT, IMG_WIDTH),
   batch_size=BATCH_SIZE
)
val_ds = tf.keras.utils.image_dataset_from_directory(
   "dataset/train",
   validation_split=0.2,
   subset="validation",
   seed=123,
   image_size=(IMG_HEIGHT, IMG_WIDTH),
   batch_size=BATCH_SIZE
)

# Obtener nombres de las clases
class_names = {i: name for i, name in enumerate(train_ds.class_names)}
print(class_names)

# Configurar datasets para rendimiento
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

# Definir el modelo CNN
num_classes = len(class_names)
model = tf.keras.Sequential([
   tf.keras.layers.Rescaling(1./255, input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)),
   tf.keras.layers.Conv2D(16, 3, padding='same', activation='relu'),
   tf.keras.layers.MaxPooling2D(),
   tf.keras.layers.Conv2D(32, 3, padding='same', activation='relu'),
   tf.keras.layers.MaxPooling2D(),
   tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
   tf.keras.layers.MaxPooling2D(),
   tf.keras.layers.Flatten(),
   tf.keras.layers.Dense(128, activation='relu'),
   tf.keras.layers.Dense(num_classes)
])

# Compilar el modelo
model.compile(
   optimizer='adam',
```

```
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy']
)

# Entrenar el modelo
epochs = 10
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs
)

# Exportar el modelo
model.save('flower_classifier.h5')

# Convertir el modelo a formato TensorFlow.js
import subprocess
subprocess.run(["mkdir", "flower_classifier_tfjs"])
subprocess.run([
    "tensorflowjs_converter",
    "--input_format", "keras",
    "flower_classifier.h5",
    "flower_classifier_tfjs"
])
with open("flower_classifier_tfjs/class_names.json", "w") as f:
    json.dump(class_names, f)
```

```
Found 2746 files belonging to 5 classes.
Using 2197 files for training.
Found 2746 files belonging to 5 classes.
Using 549 files for validation.
{0: 'daisy', 1: 'dandelion', 2: 'rose', 3: 'sunflower', 4: 'tulip'}
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/preprocessing/tf_data_layer.py:19: UserWarning: Do not pass an
  super().__init__(**kwargs)
69/69 ━━━━━━━━━━━━━━━━━━━━ 77s 1s/step - accuracy: 0.3744 - loss: 1.4506 - val_accuracy: 0.5519 - val_loss: 1.0980
Epoch 2/10
69/69 ━━━━━━━━━━━━━━━━━━━━ 68s 989ms/step - accuracy: 0.6080 - loss: 0.9972 - val_accuracy: 0.5683 - val_loss: 1.0988
Epoch 3/10
69/69 ━━━━━━━━━━━━━━━━━━━━ 87s 1s/step - accuracy: 0.6836 - loss: 0.8062 - val_accuracy: 0.5865 - val_loss: 1.0091
```

Intenté levanter el servidor en flask, pero al cambiar de colab a vscode me dio problemas.