Entrenamiento de una red neuronal con arquitectura AlexNet usando Tensorflow / Keras

Comienza a programar o generar con IA.

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
import os

# Cargar dataset CIFAR-10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Convertir etiquetas a one-hot encoding
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Parámetros
BATCH_SIZE = 32
IMAGE_SIZE = (227, 227)  # AlexNet espera 227x227

# Función de preprocesamiento (redimensiona imágenes y normaliza)
def preprocess_image(image, label):
    image = tf.image.resize(image, IMAGE_SIZE) / 255.0  # Normalización
    return image, label

# Crear datasets con `tf.data.Dataset`
train_dataset = (
    tf.data.Dataset.from_tensor_slices((x_train, y_train))
    .map(preprocess_image, num_parallel_calls=tf.data.experimental.AUTOT
    .batch(BATCH_SIZE)
    .prefetch(tf.data.experimental.AUTOTUNE)
)

test_dataset = (
    tf.data.Dataset.from_tensor_slices((x_test, y_test))
    .map(preprocess_image, num_parallel_calls=tf.data.experimental.AUTOT
    .batch(BATCH_SIZE)
    .prefetch(tf.data.experimental.AUTOTUNE)
)


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Den

def create_alexnet():
    model = Sequential([
        # Capa 1: Conv + ReLU + Pool
        Conv2D(96, (11, 11), strides=4, padding="same", input_shape=(2
        ReLU(),
        MaxPooling2D(pool_size=(3, 3), strides=2),

        # Capa 2: Conv + ReLU + Pool
        Conv2D(256, (5, 5), padding="same"),
        ReLU(),
        MaxPooling2D(pool_size=(3, 3), strides=2),
```

NotFoundError ✕ •••

Explica el error:

```
NotFoundError: Graph execution error:

Detected at node StatefulPartitionedCall
  File "<frozen runpy>", line 198, in _ru

  File "<frozen runpy>", line 88, in _run

  File "/usr/local/lib/python3.11/dist-pa

  File "/usr/local/lib/python3.11/dist-pa

  File "/usr/local/lib/python3.11/dist-pa

  File "/usr/local/lib/python3.11/dist-pa

  File "/usr/lib/python3.11/asyncio/base_

  File "/usr/lib/python3.11/asyncio/base_

  File "/usr/lib/python3.11/asyncio/event

  File "/usr/local/lib/python3.11/dist-pa

  File "/usr/local/lib/python3.11/dist-pa

  File "/usr/local/lib/python3.11/dist-pa

  File "/usr/local/lib/python3.11/dist-pa

  File "/usr/local/lib/python3.11/dist-pa

  File "/usr/local/lib/python3.11/dist-pa

  File "/usr/local/lib/python3.11/dist-pa

  File "/usr/local/lib/python3.11/dist-pa

  File "/usr/local/lib/python3.11/dist-pa

  File "/usr/local/lib/python3.11/dist-pa

  File "/usr/local/lib/python3.11/dist-pa

  File "/usr/local/lib/python3.11/dist-pa

  File "<ipython-input-4-fc452574286e>",

  File "/usr/local/lib/python3.11/dist-pa
```

```
        # Capas 3, 4, 5: Conv + ReLU
        Conv2D(384, (3, 3), padding="same"),
        ReLU(),
        Conv2D(384, (3, 3), padding="same"),
        ReLU(),
        Conv2D(256, (3, 3), padding="same"),
        ReLU(),
        MaxPooling2D(pool_size=(3, 3), strides=2),

        # Aplanar
        Flatten(),

        # Capas densas
        Dense(4096),
        ReLU(),
        Dropout(0.5),
        Dense(4096),
        ReLU(),
        Dropout(0.5),

        # Capa de salida con 10 clases (CIFAR-10)
        Dense(10, activation="softmax")
    ])
    return model

# Ruta del archivo de pesos
#WEIGHTS_PATH = "alexnet_weights.h5"
WEIGHTS_PATH = "alexnet_weights.weights.h5"

# Crear modelo
model = create_alexnet()

# Verificar si existe un modelo preentrenado
if os.path.exists(WEIGHTS_PATH):
    print("🔹 Cargando pesos preentrenados...")
    model.load_weights(WEIGHTS_PATH)
else:
    print("🔹 Entrenando desde cero...")
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convoluti
  super().__init__(activity_regularizer=activity_regularizer, **kw
🔹 Cargando pesos preentrenados...
```

```
# Compilar modelo
model.compile(optimizer="adam", loss="categorical_crossentropy", metri

# Entrenar solo si no hay pesos preentrenados
if not os.path.exists(WEIGHTS_PATH):
    model.fit(train_dataset, epochs=15, validation_data=test_dataset)

    # Guardar pesos entrenados
    model.save_weights(WEIGHTS_PATH)
    print(f"✅ Pesos guardados en {WEIGHTS_PATH}")


import numpy as np
np.argmax(model.predict(test_dataset)[0])
```

```
40/40 ━━━━━━━━━━━━━━━━━━━━ 7s 164ms/step
7
```

```
  File "/usr/local/lib/python3.11/dist-pa

  File "/usr/local/lib/python3.11/dist-pa

  File "/usr/local/lib/python3.11/dist-pa

could not find registered transfer manage
     [[{{node StatefulPartitionedCall}}]]
```

Hypothesis

The error message "NotFoundError: could not find registered transfer manager for platform Host -- check target linkage" usually indicates an issue with TensorFlow's internal communication between devices, specifically between the host (CPU) and the device where the computation is supposed to happen (GPU or TPU).

In this case, it's likely that:

1. The pre-trained weights file (alexnet_weights.weights.h5) was saved in an environment with different hardware (CPU, GPU, TPU) or TensorFlow version than the current environment. This can cause compatibility issues.
2. The path to the weights file is incorrect, or the file is corrupted.

Suggested Changes

```
# Ruta del archivo de pesos
WEIGHTS_PATH = "alexnet_weights.h5"  # (
#WEIGHTS_PATH = "alexnet_weights.weight:

# Crear modelo
model = create_alexnet()

# Verificar si existe un modelo preentre
if os.path.exists(WEIGHTS_PATH):
    print("🔹 Cargando pesos preentrena
    model.load_weights(WEIGHTS_PATH) # I
else:
    print("🔹 Entrenando desde cero..."
```

Use code with caution

Explanation of changes:

```
import pandas as pd

pd.values(y_test)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-8-a789fe06a185> in <cell line: 0>()
      1 import pandas as pd
      2
----> 3 pd.values(y_test)
```

Próximos pasos: Explicar error

```
y_test
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.]])
```

```
# Ruta del archivo de pesos
WEIGHTS_PATH = "alexnet_weights.h5"  # Changed file extension to .h5
#WEIGHTS_PATH = "alexnet_weights.weights.h5"

# Crear modelo
model = create_alexnet()

# Verificar si existe un modelo preentrenado
if os.path.exists(WEIGHTS_PATH):
    print(" ◆ Cargando pesos preentrenados...")
    model.load_weights(WEIGHTS_PATH) # Loads weights from the file if
else:
    print(" ◆ Entrenando desde cero...")
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convoluti
  super().__init__(activity_regularizer=activity_regularizer, **kw
◆ Entrenando desde cero...
```

```
# Cargar dataset CIFAR-10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
x_test[0]
```

ndarray (32, 32, 3) show data

Comienza a programar o generar con IA.

```
categories = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog',
```

1. **Change the file extension:** The original code attempts to load weights from a file named "alexnet_weights.weights.h5". This is an unusual naming convention for Keras/TensorFlow weight files. Typically, the file extension is simply ".h5". By changing `WEIGHTS_PATH` to "alexnet_weights.h5", we can ensure the code looks for the correct file extension which is commonly used for saving model weights in Keras.
2. **Ensure the weights file exists:** Double-check that the weights file ("alexnet_weights.h5") actually exists in the specified location. If you are using pre-trained weights, make sure you have downloaded the file and placed it in the correct directory.
3. **Save with the correct extension:** When saving the weights, make sure to use the ".h5" extension:

```
model.save_weights(WEIGHTS_PATH)
```

Use code with caution

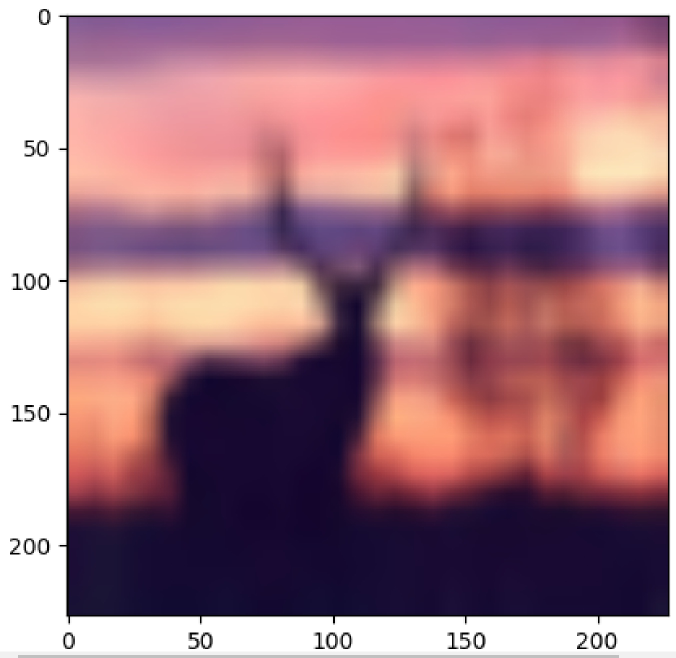
By making these changes, we can address the potential issues related to file paths and extensions. After making the changes above and executing the code, if the error still shows up it is due to a hardware conflict of your environment with the hardware where the weights were originally saved. You should train the model from scratch in this case.

Califica esta respuesta

```
im=tf.image.resize(x_test[211], IMAGE_SIZE) / 255.0
```

```
import matplotlib.pyplot as plt
plt.imshow(im)
```

```
<matplotlib.image.AxesImage at 0x7a0aeadf8350>
```



```
np.argmax(model.predict(np.expand_dims(im, axis=0)))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
9
```

```
x_test[3]
```

```
ndarray (32, 32, 3) show data
```

```
x_test[11]
```

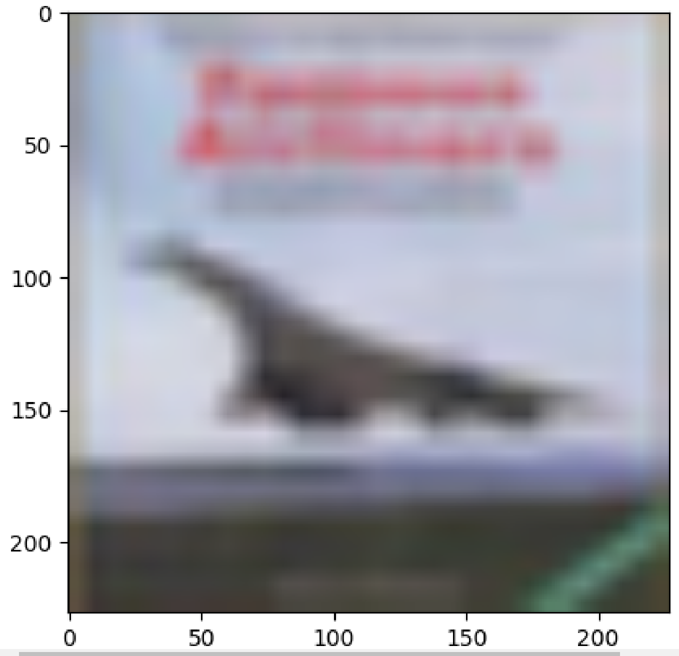
```
ndarray (32, 32, 3) show data
```

Comienza a programar o generar con IA.

```
im=tf.image.resize(x_test[3], IMAGE_SIZE) / 255.0
```

```
import matplotlib.pyplot as plt
plt.imshow(im)
```

```
<matplotlib.image.AxesImage at 0x7dac3422e410>
```



```
np.argmax(model.predict(np.expand_dims(im, axis=0)))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
0
```

```
import numpy as np
categories[np.argmax(model.predict(np.expand_dims(im, axis=0)))]
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
```

```
# Esto es para agregar un corchete. Importante, debe ser array o tensor,
import numpy as np
categories[np.argmax(model.predict(np.array([im])))]
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
```

```
im
```

```
<tf.Tensor: shape=(227, 227, 3), dtype=float32, numpy=
array([[[0.5294118 , 0.37254903, 0.5921569 ],
        [0.5294118 , 0.37254903, 0.5921569 ],
        [0.5294118 , 0.37254903, 0.5921569 ],
        ...,
        [0.46666667, 0.2627451 , 0.44313726],
        [0.46666667, 0.2627451 , 0.44313726],
        [0.46666667, 0.2627451 , 0.44313726]],

       [[0.5294118 , 0.37254903, 0.5921569 ],
        [0.5294118 , 0.37254903, 0.5921569 ],
        [0.5294118 , 0.37254903, 0.5921569 ],
        ...,
        [0.46666667, 0.2627451 , 0.44313726],
        [0.46666667, 0.2627451 , 0.44313726],
        [0.46666667, 0.2627451 , 0.44313726]],

       [[0.5294118 , 0.37254903, 0.5921569 ],
             [0.5294118 , 0.37254903, 0.5921569 ],
             [0.5294118 , 0.37254903, 0.5921569 ],
             ...,
             [0.46666667, 0.2627451 , 0.44313726],
             [0.46666667, 0.2627451 , 0.44313726],
             [0.46666667, 0.2627451 , 0.44313726]],

            ...,

            [[0.09803922, 0.07058824, 0.20784314],
             [0.09803922, 0.07058824, 0.20784314],
             [0.09803922, 0.07058824, 0.20784314],
             ...,
             [0.10980392, 0.07843138, 0.21568628],
             [0.10980392, 0.07843138, 0.21568628],
             [0.10980392, 0.07843138, 0.21568628]],

            [[0.09803922, 0.07058824, 0.20784314],
             [0.09803922, 0.07058824, 0.20784314],
             [0.09803922, 0.07058824, 0.20784314],
             ...,
             [0.10980392, 0.07843138, 0.21568628],
             [0.10980392, 0.07843138, 0.21568628],
             [0.10980392, 0.07843138, 0.21568628]],

            [[0.09803922, 0.07058824, 0.20784314],
             [0.09803922, 0.07058824, 0.20784314],
             [0.09803922, 0.07058824, 0.20784314],
             ...,
             [0.10980392, 0.07843138, 0.21568628],
             [0.10980392, 0.07843138, 0.21568628],
             [0.10980392, 0.07843138, 0.21568628]]], dtype=float32)>
```

```
imexp=np.expand_dims(im, axis=0)
```

```
imexp.shape
```

```
(1, 227, 227, 3)
```