```python
import os
import numpy as np
import tensorflow as tf
import json
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten,
Dense, Concatenate
from sklearn.model_selection import train_test_split

# Configuración
image_dir = r"C:\Users\ismae\Desktop\TFG\Python Windows\frames"  # Ruta a
las imágenes originales
mask_dir = r"C:\Users\ismae\Desktop\TFG\Mascaras\entrenamiento"  # Ruta a
las máscaras
labels_file =
r"C:\Users\ismae\Desktop\22Noviembre\DATA_22_Noviembre\DATA-Tendon-
Long\file_info.json"  # Archivo JSON con las puntuaciones de cada imagen
img_size = (256, 256)  # Tamaño al que redimensionaremos las imágenes
batch_size = 16
epochs = 50

# Cargar las puntuaciones desde el archivo JSON
with open(labels_file, 'r') as file:
    labels_data = json.load(file)

# Función para cargar imágenes, máscaras y sus puntuaciones
def load_data(image_dir, mask_dir, labels_data):
    images = []
    masks = []
    labels = []

    # Listar todas las imágenes disponibles
    image_files = os.listdir(image_dir)
    mask_files = os.listdir(mask_dir)

    for img_file in image_files:
        base_name = os.path.splitext(img_file)[0]  # Extraer nombre base
sin extensión
        mask_file = f"{base_name}_mask.png"

        # Comparar sin importar mayúsculas/minúsculas
        if mask_file.lower() in [m.lower() for m in mask_files]:

            # Cargar imagen y máscara
            img = load_img(os.path.join(image_dir, img_file),
target_size=img_size)
            mask = load_img(os.path.join(mask_dir, mask_file),
target_size=img_size, color_mode='grayscale')
```

```
            # Convertir a arrays y normalizar
            img = img_to_array(img) / 255.0
            mask = img_to_array(mask) / 255.0

            # Buscar la etiqueta correspondiente en el JSON
            for entry in labels_data:
                if os.path.basename(entry["Original_image"]) ==
f"{base_name}.jpg":
                    # Extraer las puntuaciones como lista de floats
                    score = [float(entry["answers"]["tissue"]),
float(entry["answers"]["morphology"]),
                             float(entry["answers"]["upper border"]),
float(entry["answers"]["lower border"]),
                             float(entry["answers"]["bone"])]
                    images.append(img)
                    masks.append(mask)
                    labels.append(score)

    print(f"Total de imágenes cargadas: {len(images)}")
    print(f"Total de máscaras cargadas: {len(masks)}")
    print(f"Total de etiquetas cargadas: {len(labels)}")

    return np.array(images), np.array(masks), np.array(labels)



# Cargar todos los datos
images, masks, labels = load_data(image_dir, mask_dir, labels_data)

# Dividir los datos en entrenamiento (80%) y validación (20%)
train_images, val_images, train_masks, val_masks, train_labels,
val_labels = train_test_split(
    images, masks, labels, test_size=0.2, random_state=42
)

# Definir el modelo
def build_model(input_shape):
    # Entrada de la imagen
    img_input = Input(shape=input_shape)

    # Entrada de la máscara
    mask_input = Input(shape=(input_shape[0], input_shape[1], 1))

    # Concatenar imagen y máscara
    x = Concatenate()([img_input, mask_input])

    # Capas convolucionales
    x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
```

```
    x = MaxPooling2D((2, 2))(x)

    x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2))(x)

    x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2))(x)

    x = Flatten()(x)

    # Salida: 5 puntuaciones
    output = Dense(5, activation='linear')(x)

    model = Model(inputs=[img_input, mask_input], outputs=output)
    model.compile(optimizer='adam', loss='mse', metrics=['mae'])

    return model

# Construir el modelo
model = build_model((img_size[0], img_size[1], 3))

# Entrenar el modelo
history = model.fit(
    [train_images, train_masks], train_labels,
    validation_data=([val_images, val_masks], val_labels),
    batch_size=batch_size,
    epochs=epochs
)

# Guardar el modelo entrenado
model.save(r"C:\Users\ismae\Desktop\TFG\modelos_puntuacion\modelo_puntuac
ion1.h5")

print("Entrenamiento completado y modelo guardado.")
```