```python
import os
import numpy as np
import tensorflow as tf
import json
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import load_img, img_to_array,
ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten,
Dense, Concatenate, Dropout, BatchNormalization
from tensorflow.keras.callbacks import Callback, ModelCheckpoint,
ReduceLROnPlateau
from sklearn.model_selection import train_test_split

# Configuración
image_dir = r"C:\Users\ismae\Desktop\TFG\Python Windows\frames"
mask_dir = r"C:\Users\ismae\Desktop\TFG\Mascaras\entrenamiento"
labels_file =
r"C:\Users\ismae\Desktop\22Noviembre\DATA_22_Noviembre\DATA-Tendon-
Long\file_info.json"
model_dir = r"C:\Users\ismae\Desktop\TFG\modelos_puntuacion"
img_size = (128, 128)
batch_size = 16
epochs = 50

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

# Función para cargar los datos
def load_data(image_dir, mask_dir, labels_data):
    images, masks, labels = [], [], []
    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]
        mask_file = f"{base_name}_mask.png"
        if mask_file.lower() in [m.lower() for m in mask_files]:
            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')
            img = img_to_array(img) / 255.0
            mask = img_to_array(mask) / 255.0
            for entry in labels_data:
                if os.path.basename(entry["Original_image"]) ==
f"{base_name}.jpg":
```

```
                    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)}")
    return np.array(images), np.array(masks), np.array(labels)

# Cargar los datos
images, masks, labels = load_data(image_dir, mask_dir, labels_data)
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):
    img_input = Input(shape=input_shape)
    mask_input = Input(shape=(input_shape[0], input_shape[1], 1))
    x = Concatenate()([img_input, mask_input])

    # Bloque 1
    x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
    x = BatchNormalization()(x)
    x = MaxPooling2D((2, 2))(x)
    x = Dropout(0.25)(x)

    # Bloque 2
    x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = BatchNormalization()(x)
    x = MaxPooling2D((2, 2))(x)
    x = Dropout(0.25)(x)

    # Bloque 3
    x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
    x = BatchNormalization()(x)
    x = MaxPooling2D((2, 2))(x)
    x = Dropout(0.25)(x)

    # Bloque 4
    x = Conv2D(256, (3, 3), activation='relu', padding='same')(x)
    x = BatchNormalization()(x)
    x = MaxPooling2D((2, 2))(x)
    x = Dropout(0.25)(x)
```

```
    x = Flatten()(x)
    output = Dense(5, activation='linear')(x)  # Una salida para cada
sección
    model = Model(inputs=[img_input, mask_input], outputs=output)
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001)
, loss='mse', metrics=['mae'])
    return model

model = build_model((img_size[0], img_size[1], 3))

# Callback para monitorizar el error
class ValidationErrorMonitor(Callback):
    def on_epoch_end(self, epoch, logs=None):
        predictions = self.model.predict([val_images, val_masks])
        errors = np.abs(predictions - val_labels)
        mean_errors = errors.mean(axis=0)
        total_mean_error = mean_errors.mean()
        print(f"\nEpoch {epoch + 1} - Error medio por sección
(validación):")
        print(f"  Tissue: {mean_errors[0]:.4f}, Morphology:
{mean_errors[1]:.4f}, "
              f"Upper Border: {mean_errors[2]:.4f}, Lower Border:
{mean_errors[3]:.4f}, "
              f"Bone: {mean_errors[4]:.4f}")
        print(f"  Error medio total (validación):
{total_mean_error:.4f}\n")

# Callbacks
checkpoint_callback = ModelCheckpoint(
    filepath=os.path.join(model_dir, "modelo_puntuacion_best.h5"),
    monitor='val_loss',
    save_best_only=True,
    mode='min',
    verbose=1
)

reduce_lr_callback = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=5,
    verbose=1,
    mode='min'
)

# Generador de datos con aumento
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
```

```
    zoom_range=0.1,
    horizontal_flip=True
)

# Entrenar el modelo
history = model.fit(
    datagen.flow([train_images, train_masks], train_labels,
batch_size=batch_size),
    validation_data=([val_images, val_masks], val_labels),
    epochs=epochs,
    callbacks=[ValidationErrorMonitor(), checkpoint_callback,
reduce_lr_callback]
)

# Guardar el modelo final
model.save(os.path.join(model_dir, "modelo_puntuacion_final.h5"))

# Graficar la historia del entrenamiento
def plot_training_history(history):
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    mae = history.history['mae']
    val_mae = history.history['val_mae']

    epochs_range = range(1, len(loss) + 1)

    plt.figure(figsize=(14, 5))

    # Gráfica de la pérdida
    plt.subplot(1, 2, 1)
    """.plot(epochs_range, loss, label='Pérdida - Entrenamiento',
color='blue')"""
    plt.plot(epochs_range, val_loss, label='Pérdida - Validación',
color='orange')
    plt.title('Evolución de la Pérdida')
    plt.xlabel('Épocas')
    plt.ylabel('Pérdida')
    plt.legend()
    plt.grid()

    # Gráfica de MAE
    plt.subplot(1, 2, 2)
    """plt.plot(epochs_range, mae, label='MAE - Entrenamiento',
color='green')"""
    plt.plot(epochs_range, val_mae, label='MAE - Validación',
color='red')
    plt.title('Evolución de MAE')
    plt.xlabel('Épocas')
    plt.ylabel('MAE')
```

```
    plt.legend()
    plt.grid()

    plt.tight_layout()
    plt.show()

plot_training_history(history)

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