Tecnológico Nacional de México Campus Culiacán

TECNOLÓGICO NACIONAL DE MÉXICO

"Tarea 1"

Nombre de los alumnos:

Rios Sauceda Jose Lorenzo.

Cazarez Ibarra Francisco Javier

Docente: *Zuriel Dathan Mora Félix.*

Materia: *Inteligencia Artificial.*

Carrera: *Ing. En sistemas computacionales.*

Semestre: *8*

Horario: *9-10AM*

Fecha: *24 de mayo del 2025.*

Archivo config.py

Se importan las bibliotecas opencv para el clasificador haar y os para el manejo de las rutas de nuestros dataset y demás, el parámetro TAMANO_IMG define el tamaño final que tendrán las imágenes de nuestro dataset después de ser procesadas.

Modelo > config.py > ...

```python
import cv2
import os

#Rutas de los dataset usados
RUTA_ORIGINAL = r"C:\Users\loren\Desktop\Semestre 8\Inteligencia artificial\Pre-procesamiento\Dataset Imagenes Emociones"
RUTA_SALIDA_PROCESADA = r"C:\Users\loren\Desktop\Semestre 8\Inteligencia artificial\Pre-procesamiento\Imagenes Procesadas"
RUTA_HAAR_CASCADE = cv2.data.haarcascades + 'haarcascade_frontalface_default.xml'

#Parámetros de Imagen
TAMANO_IMG = (224, 224)  # Tamaño al que se redimensionarán las imágenes (usado en  elmodelo  y preprocesamiento)
```

Los primeros cuatro parámetros se utilizan en para crear variaciones artificiales de las imágenes en nuestro dataset (rotación, zoom, y iluminaciones). Despues están los valores que definen como será entrenado el modelo, se define la cantidad de imágenes procesadas en cada paso, pasadas completas por el dataset, tasa de aprendizaje y semillas para procesos aleatorios para aumentos de datos o particiones del dataset

```python
#Parametros de aumento de datos
GRADOS_ROTACION_MAX = 15
FACTOR_ZOOM = 0.1
FACTOR_ILUMINACION_MIN = 0.6
FACTOR_ILUMINACION_MAX = 1.4

#Valores para entrenamiento
BATCH_SIZE = 32
EPOCHS = 20  # Aumentado para mejor aprendizaje
LEARNING_RATE = 0.0001
SEED = 123  # Para reproducibilidad

# División de dataset en porcentajes (porcentajes en decimales)
FRAC_TRAIN = 0.70
FRAC_VAL = 0.15
FRAC_TEST = 0.15
```

Por ultimo tenemos como se divide el conjunto de imágenes del dataset, que van en 70% dedicado para el entrenamiento del modelo, 15% para evaluar el rendimiento mientras se entrena y el otro 15% para evaluar rendimiento después del entrenamiento. Por ultimo se tiene el nombre con el que se guarda el modelo entrenado, esta en formato .keras para que sea compatible con tensorflow y keras.

```
# División de dataset en porcentajes (porcentajes en decimales)
FRAC_TRAIN = 0.70
FRAC_VAL = 0.15
FRAC_TEST = 0.15

#Ruta donde se guardara el dataset procesado
NOMBRE_MODELO_GUARDADO = "modelo_reconocimiento_emociones_ResNet50.keras"
```

Archivo Preprocesamiento.py

Se importan las bibliotecas y las variables anteriormente definidas en el archivo anteriormente documenado.

```
import os
import cv2
import numpy as np

from config import RUTA_ORIGINAL, RUTA_SALIDA_PROCESADA, TAMANO_IMG, \
                   GRADOS_ROTACION_MAX, FACTOR_ZOOM, \
                   FACTOR_ILUMINACION_MIN, FACTOR_ILUMINACION_MAX
```

Esta función aplica un cambio aleatorio en el brillo/iluminación de la imagen, multiplicando los pixeles por un valor aleatorio.

```python
def ajustar_iluminacion(imagen):
    # Los factores de iluminación se obtienen de config.py
    factor = np.random.uniform(FACTOR_ILUMINACION_MIN, FACTOR_ILUMINACION_MAX)
    return np.clip(imagen * factor, 0, 255).astype(np.uint8)
```

Esta función rota la imagen en un angulo aleatorio, cv2.warpaffine para rotar y evita los bordes negros con cv2.border_reflect

```python
def rotar_imagen(imagen):
    h, w = imagen.shape[:2]
    # Los grados máximos de rotación se obtienen de config.py
    angulo = np.random.uniform(-GRADOS_ROTACION_MAX, GRADOS_ROTACION_MAX)
    M = cv2.getRotationMatrix2D((w//2, h//2), angulo, 1)
    return cv2.warpAffine(imagen, M, (w, h), borderMode=cv2.BORDER_REFLECT)
```

Esta función realiza un zoom aleatorio (puede ser alejamiento o acercamiento), si se amplia recorta el centro y si se reduce la centra en un fondo para mantener el tamaño original

```python
def escalar_imagen(imagen):
    h, w = imagen.shape[:2]
    # El factor de zoom se obtiene de config.py
    factor = np.random.uniform(1 - FACTOR_ZOOM, 1 + FACTOR_ZOOM)
    imagen_zoom = cv2.resize(imagen, (0, 0), fx=factor, fy=factor, interpolation=cv2.INTER_LINEAR)

    # Recortar o rellenar imagen para que mantenga el TAMANO_IMG
    if factor > 1:
        startx = (imagen_zoom.shape[1] - w) // 2
        starty = (imagen_zoom.shape[0] - h) // 2
        return imagen_zoom[starty:starty + h, startx:startx + w]
    else: # Si la imagen se encogió (zoom out), rellenar con ceros
        nueva = np.zeros((h, w, 3), dtype=np.uint8)
        startx = (w - imagen_zoom.shape[1]) // 2
        starty = (h - imagen_zoom.shape[0]) // 2
        nueva[starty:starty + imagen_zoom.shape[0], startx:startx + imagen_zoom.shape[1]] = imagen_zoom
        return nueva
```

Esta función convierte los valores de pixeles de 0-255 a 0-1, se hizo asi para el entrenamiento.

```python
def normalizar_pixeles(imagen):
    return imagen.astype(np.float32) / 255.0
```

Aplica las funciones anteriores de iluminación, rotación y escalado a las imágenes y redimensiona utilizando la variable TAMANO_IMG, normaliza los pixeles y al final devuelve la imagen procesada para ser usada en el entrenamiento.

```python
def preprocesar_imagen_completa(imagen):

    imagen_aumentada = ajustar_iluminacion(imagen)
    imagen_aumentada = rotar_imagen(imagen_aumentada)
    imagen_aumentada = escalar_imagen(imagen_aumentada)

    # para asegurar que todas las imágenes al fina tengan el mismo tamaño.
    imagen_final = cv2.resize(imagen_aumentada, TAMANO_IMG, interpolation=cv2.INTER_AREA)

    # Normalizar los píxeles
    imagen_normalizada = normalizar_pixeles(imagen_final)

    return imagen_normalizada
```

Por ultimo esta función empieza recorriendo las carpetas que contiene el dataset original, analiza cada imagen para convetirlas en RGB, se preprocesa y la guarda en la carpeta de salida, también ignora archivos no validos o que no se pueden leer (como imágenes corrompidas) y por ultimo imprime estadísticas de cuantas imágenes se procesaron sin problemas y cuantas fueron omitidas

```python
def procesar_dataset():

    print(f"Iniciando preprocesamiento de imágenes desde: {RUTA_ORIGINAL}")
    print(f"Las imágenes procesadas se guardarán en: {RUTA_SALIDA_PROCESADA}")

    total_imagenes_procesadas = 0
    total_imagenes_saltadas = 0

    # Crear el directorio de salida si no existe
    os.makedirs(RUTA_SALIDA_PROCESADA, exist_ok=True)

    for clase in os.listdir(RUTA_ORIGINAL):
        ruta_clase = os.path.join(RUTA_ORIGINAL, clase)
        ruta_salida_clase = os.path.join(RUTA_SALIDA_PROCESADA, clase)
        os.makedirs(ruta_salida_clase, exist_ok=True)

        if not os.path.isdir(ruta_clase):
            continue # Saltar si no es un directorio (ej. .DS_Store)

        for archivo in os.listdir(ruta_clase):
            ruta_img = os.path.join(ruta_clase, archivo)

            # Solo procesar archivos de imagen comunes
            if not archivo.lower().endswith(('.png', '.jpg', '.jpeg', '.gif', '.bmp')):
                continue
```

```python
            imagen = cv2.imread(ruta_img)
            if imagen is None:
                print(f"Error: No se pudo cargar la imagen {ruta_img}")
                total_imagenes_saltadas += 1
                continue

            imagen_rgb = cv2.cvtColor(imagen, cv2.COLOR_BGR2RGB)

            # Llamar a la nueva función de preprocesamiento
            imagen_proc = preprocesar_imagen_completa(imagen_rgb)

            # Guardar imagen preprocesada (de 0-1 a 0-255 y de RGB a BGR para guardar)
            imagen_guardar = (imagen_proc * 255).astype(np.uint8)
            salida_path = os.path.join(ruta_salida_clase, archivo)
            cv2.imwrite(salida_path, cv2.cvtColor(imagen_guardar, cv2.COLOR_RGB2BGR))
            total_imagenes_procesadas += 1

    print(f"\nPreprocesamiento terminado. {total_imagenes_procesadas} imágenes procesadas, {total_imagenes_saltadas} imágenes saltadas.")

if __name__ == "__main__":

    procesar_dataset()
```

Archivo modelo.py

Se importan las librerías y variable de el archivo config. Se define la función para construir y devolver un modelo basado en ResNet50 para clasificación. Se carga la arquitectura preentrenada de ResNET50 preentrenada en el dataset ImageNet, se configura para las imágenes RGB del tamaño de la variable TAMANO_IMG. Se congelan los pesos de ResNet50 para que no se actualicen durante el entrenamiento.

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers    Import "tensorflow.keras" could not be resolved
from config import TAMANO_IMG # Importa el tamaño de imagen de config

def create_emotion_model(num_classes):

    print(f"Creando modelo ResNet50 para {num_classes} clases...")

    # Cargar el modelo base pre-entrenado (ResNet50)
    base_model = keras.applications.ResNet50(
        input_shape=TAMANO_IMG + (3,), # (224, 224, 3) para imágenes RGB
        include_top=False, # No incluimos las capas de clasificación finales de ImageNet
        weights='imagenet' # Usamos los pesos pre-entrenados en ImageNet
    )

    # Congelar las capas del modelo base
    # Esto evita que los pesos de ResNet50 se actualicen durante el entrenamiento inicial
    base_model.trainable = False
```

Despues se pasa la imagen de entrada por la ResNet50 que se congelo para obtener sus características, se aplica una capa de poolingglobal para reducir dimensiones del mapa de características y se agrega una capa dropout para desactivar aleatoriamente el 20% de las neuronas durante el entrenamiento, esto se hizo para evitar el sobreajuste.

Luego se activa softmax para producir probabilidades mutliclase, se crea el modelo completo conectando entrada y salida con las capas definidas, al final se imprime un resumen en la consola mostrando el numero de capas y parámetros entrenables y los que no se pueden entrenar.

```python
    # Construir el nuevo modelo
    inputs = keras.Input(shape=TAMANO_IMG + (3,))

    # Pasar las entradas a través del modelo base.
    # 'training=False' es importante para que las capas de BatchNormalization del modelo base
    x = base_model(inputs, training=False)

    # Añadir una capa GlobalAveragePooling2D para aplanar las características
    x = layers.GlobalAveragePooling2D()(x)

    # Añadir una capa Dropout para regularización
    x = layers.Dropout(0.2)(x)

    # Añadir la capa de salida con 'softmax' para clasificación multiclase
    outputs = layers.Dense(num_classes, activation='softmax')(x)

    # Crear el modelo final
    model = keras.Model(inputs, outputs)

    print("Modelo ResNet50 creado:")
    model.summary()

    return model

if __name__ == "__main__":
    dummy_model = create_emotion_model(num_classes=5)
    # No se entrenará aquí, solo se mostrará el resumen.
```

Archivo entrenamiento.py

Se importan las bibliotecas y variables de config.py además de la función que construye el modelo.

Modelo > Entrenamiento.py > load_full_dataset

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers    Import "tensorflow.keras" could not be resolved
import numpy as np
import os
from sklearn.model_selection import train_test_split

from config import RUTA_SALIDA_PROCESADA, TAMANO_IMG, BATCH_SIZE, \
                   EPOCHS, LEARNING_RATE, SEED, NOMBRE_MODELO_GUARDADO
from Modelo import create_emotion_model
```

Esta función carga el dataset ya preprocesado sin separarlas aun en entrenamiento, validación y prueba. Cada imagen se convierte en una matriz NumPy y se etiqueta también.

```python
def load_full_dataset():
    print("Cargando dataset completo sin división...")
    full_ds = keras.preprocessing.image_dataset_from_directory(
        RUTA_SALIDA_PROCESADA,
        image_size=TAMANO_IMG,
        batch_size=1,  # batch=1 para cargar todas las imágenes individualmente
        shuffle=True,
        seed=SEED,
        label_mode='categorical'
    )

    images = []
    labels = []

    for img, label in full_ds:
        images.append(img[0].numpy())
        labels.append(label[0].numpy())

    images = np.array(images)
    labels = np.array(labels)
    print(f"Dataset cargado con {len(images)} imágenes.")
    return images, labels, full_ds.class_names
```

Aquí se divide el conjunto de datos en 3 partes, 70% para el entrenamiento, 15% para la validación y 15% para pruebas. Se uso stratify para que la distribución de las clases sea la misma en cada subconjunto.

```python
def split_dataset(images, labels):
    # Dividir en train (70%) y temp (30%)
    X_train, X_temp, y_train, y_temp = train_test_split(
        images, labels, test_size=0.3, random_state=SEED, stratify=labels
    )
    # Dividir temp en val (15%) y test (15%)
    X_val, X_test, y_val, y_test = train_test_split(
        X_temp, y_temp, test_size=0.5, random_state=SEED, stratify=y_temp
    )
    print(f"Split de datos: Train={len(X_train)}, Val={len(X_val)}, Test={len(X_test)}")
    return (X_train, y_train), (X_val, y_val), (X_test, y_test)
```

Esta función convierte los arrays de las imágenes y etiquetas en datasets de TensorFlow usando tf.data.dataset, si se cumple que augment=true entonces se aplican las transformaciones aleatorias vistas anteriormente (zoom, rotación, brillo)

```
def prepare_tf_dataset(images, labels, augment=False):
    ds = tf.data.Dataset.from_tensor_slices((images, labels))
    ds = ds.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
    if augment:
        augmentation = keras.Sequential([
            layers.RandomFlip("horizontal"),
            layers.RandomRotation(0.1),
            layers.RandomZoom(0.1),
            layers.RandomBrightness(0.2)
        ])
        ds = ds.map(lambda x, y: (augmentation(x, training=True), y), num_parallel_calls=tf.data.AUTOTUNE)
    return ds
```

Se compila el modelo utilizando la optimización adam, la función de perdida y se mide la precisión para entrenar el modelo con los dataset de entrenamiento y validación.

```
def compile_and_train_model(model, train_ds, val_ds):
    print("\nCompilando modelo...")
    optimizer = keras.optimizers.Adam(learning_rate=LEARNING_RATE)
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    print("Entrenando modelo...")
    history = model.fit(train_ds,
                        epochs=EPOCHS,
                        validation_data=val_ds)
    print("Entrenamiento finalizado.")
    return history
```

Esta función guarda el modelo entrenado en formato TensorFlow. Al final de todo el modelo evallua en el conjunto de pruebas y muestra la precisión que se pudo obtener.

```python
if __name__ == "__main__":
    images, labels, class_names = load_full_dataset()
    num_classes = len(class_names)
    (X_train, y_train), (X_val, y_val), (X_test, y_test) = split_dataset(images, labels)

    train_ds = prepare_tf_dataset(X_train, y_train, augment=True)
    val_ds = prepare_tf_dataset(X_val, y_val, augment=False)
    test_ds = prepare_tf_dataset(X_test, y_test, augment=False)

    model = create_emotion_model(num_classes)
    history = compile_and_train_model(model, train_ds, val_ds)
    save_model(model, NOMBRE_MODELO_GUARDADO)

    # Opcional: evaluar en test set
    print("\nEvaluando modelo en test set...")
    test_loss, test_acc = model.evaluate(test_ds)
    print(f"Test Accuracy: {test_acc*100:.2f}%")
```

Archivo evaluación.py

Se importan las librerías y constantes necesarias desde config.py

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing import image_dataset_from_directory    Impc
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import os

from config import RUTA_SALIDA_PROCESADA, TAMANO_IMG, BATCH_SIZE, SEED, \
                   NOMBRE_MODELO_GUARDADO
```

En esta función se carga el subconjunto de validación desde la ubicación de las imágenes procesadas, se usa validation_split para obtener el 20% de los datos como conjunto de pruebas para al final devolver el dataset y las etiquetas.

```python
def load_test_dataset():
    print("Cargando el dataset de prueba (o validación si no hay test set dedicado)...")
    test_ds = image_dataset_from_directory(
        RUTA_SALIDA_PROCESADA,
        validation_split=0.2, # Usamos la misma lógica para obtener el subconjunto de validación
        subset="validation",
        seed=SEED,
        image_size=TAMANO_IMG,
        batch_size=BATCH_SIZE,
        label_mode='categorical'
    )

    class_names = test_ds.class_names
    test_ds = test_ds.cache().prefetch(buffer_size=tf.data.AUTOTUNE)
    print("Dataset de prueba/validación cargado y optimizado.")

    return test_ds, class_names
```

Se carga el modelo en formato keras y muestra mensajes en caso de error o éxito, devuelve el modelo cargado o none si hubo algún error.

```python
def load_trained_model(model_path):
    print(f"Cargando modelo desde: {model_path}")
    try:
        model = keras.models.load_model(model_path)
        print("Modelo cargado exitosamente.")
        return model
    except Exception as e:
        print(f"Error al cargar el modelo: {e}")
        return None
```

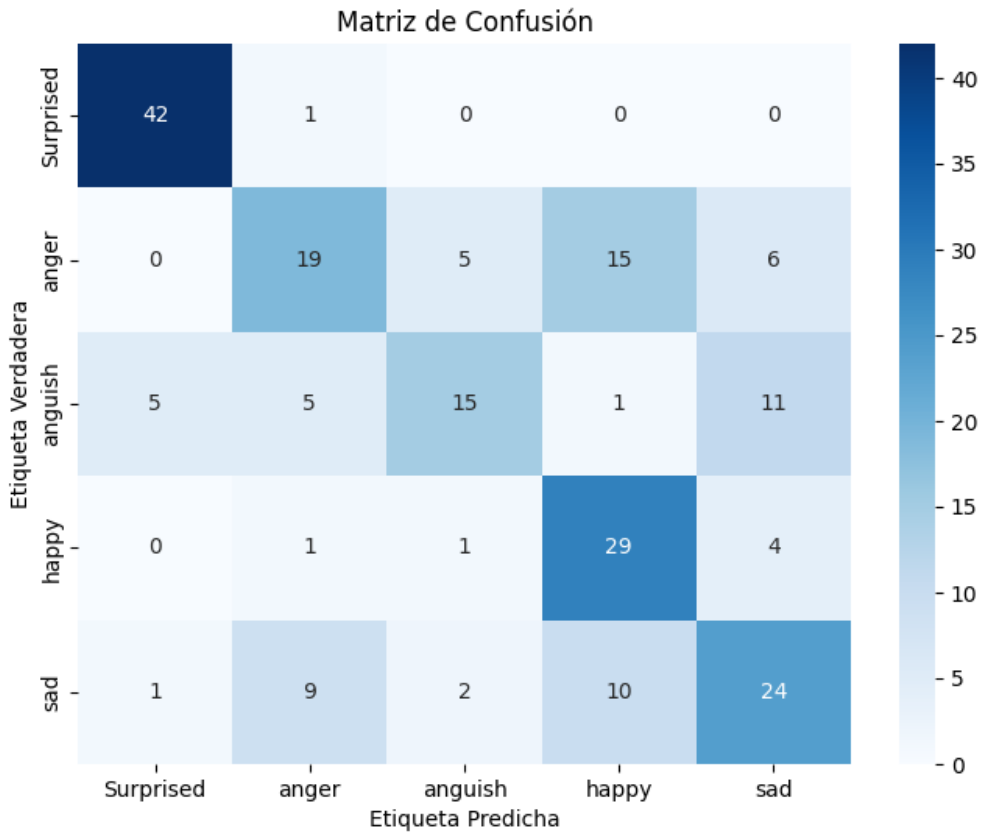
En esta función comenzamos evaluando el modelo con model.evaluate y se extraen las etiquetas true y las predicciones del dataset, para calcular la matriz de confusión y generar un reporte de clasificación. Se grafica la matriz de confusión con ayuda de seaborn.

```python
def evaluate_and_report(model, test_ds, class_names):
    print("\nRealizando evaluación final en el conjunto de prueba/validación...")
    loss, accuracy = model.evaluate(test_ds)

    print(f"Pérdida en el conjunto de prueba/validación: {loss:.4f}")
    print(f"Precisión en el conjunto de prueba/validación: {accuracy:.4f}")

    y_true = np.concatenate([y.numpy() for x, y in test_ds], axis=0)
    y_true_labels = np.argmax(y_true, axis=1)

    y_pred_probs = model.predict(test_ds)
    y_pred_labels = np.argmax(y_pred_probs, axis=1)

    print("\nMatriz de Confusión:")
    cm = confusion_matrix(y_true_labels, y_pred_labels)
    print(cm)

    print("\nReporte de Clasificación:")
    print(classification_report(y_true_labels, y_pred_labels, target_names=class_names))

    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
    plt.xlabel("Etiqueta Predicha")
    plt.ylabel("Etiqueta Verdadera")
    plt.title("Matriz de Confusión")
    plt.show()
```

Esta función recibe history producto del entrenamiento y genera dos graficas, la precisión del entrenamiento y validación y la perdida en el entrenamiento y validación.

```python
Click to add a breakpoint
def plot_training_history(history):
    if history is None:
        print("No hay historial de entrenamiento para graficar.")
        return

    print("\nGenerando gráficas de precisión y pérdida del historial de entrenamiento...")
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Precisión de Entrenamiento')
    plt.plot(history.history['val_accuracy'], label='Precisión de Validación')
    plt.xlabel('Época')
    plt.ylabel('Precisión')
    plt.title('Precisión del Modelo durante el Entrenamiento')
    plt.legend()

    plt.subplot(1, 2, 2)
    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 del Modelo durante el Entrenamiento')
    plt.legend()
    plt.show()
```

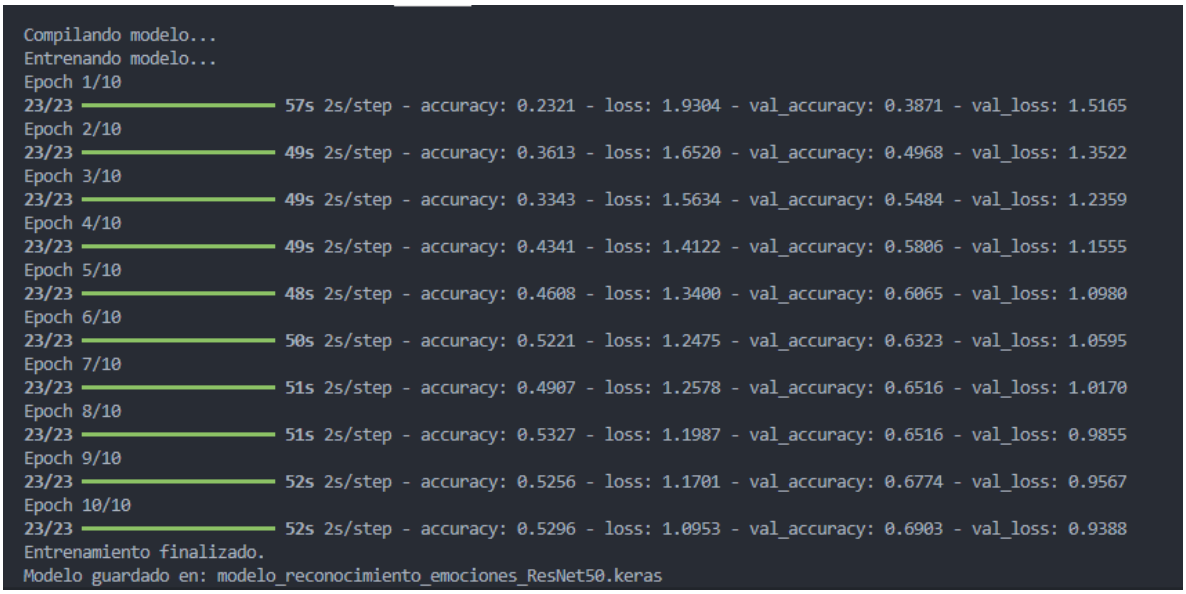
Capturas de ejecuciones:



Consola de ejecución de **Evaluacion.py**

```
E: End of sequence
7/7 ━━━━━━━━━━━━━━━━━━━━ 15s 2s/step

Matriz de Confusión:
[[42  1  0  0  0]
 [ 0 19  5 15  6]
 [ 5  5 15  1 11]
 [ 0  1  1 29  4]
 [ 1  9  2 10 24]]

Reporte de Clasificación:
              precision    recall  f1-score   support

   Surprised       0.88      0.98      0.92        43
       anger       0.54      0.42      0.47        45
     anguish       0.65      0.41      0.50        37
       happy       0.53      0.83      0.64        35
         sad       0.53      0.52      0.53        46

    accuracy                           0.63       206
   macro avg       0.63      0.63      0.61       206
weighted avg       0.63      0.63      0.61       206
```

Captura de **Entrenamiento.py**



Justificación de las Decisiones Metodológicas

En el desarrollo de este proyecto de reconocimiento de emociones en rostros humanos, cada etapa, desde el pre-procesamiento hasta la evaluación, ha sido cuidadosamente seleccionada y justificada para maximizar el rendimiento y la robustez del modelo. A continuación, se detallan las razones detrás de las principales decisiones metodológicas:

1. Método de Pre-procesamiento

Inicialmente, se consideró la detección de rostros para centrar el modelo en la región de interés. Sin embargo, se optó por un enfoque que procesa la **imagen completa** con redimensionamiento y aumento de datos por las siguientes razones:

- **Simplicidad y Robustez Operacional:** Eliminar la fase de detección facial simplifica el pipeline de pre-

procesamiento. Esto evita posibles fallos en la detección de rostros (ej. rostros no detectados, múltiples rostros detectados incorrectamente, o detección de objetos no faciales), que podrían resultar en imágenes descartadas o ruido en el dataset, haciendo el proceso más robusto para diferentes tipos de imágenes de entrada.

- **Aumento de Datos (Data Augmentation):** Se implementaron técnicas de aumento de datos como ajuste de iluminación, rotación aleatoria y escalado (zoom). Estas transformaciones se aplican a cada imagen procesada, lo cual es crucial por varios motivos:
    - **Variabilidad del Dataset:** Al generar variaciones artificiales de las imágenes existentes, se incrementa la diversidad del conjunto de datos. Esto es vital para entrenar un modelo que pueda generalizar bien a nuevas imágenes con diferentes condiciones de iluminación, ángulos y tamaños.
    - **Prevención del Sobreajuste (Overfitting):** Un dataset más variado ayuda a que el modelo no memorice los ejemplos de entrenamiento, sino que aprenda características más generales y robustas, reduciendo la probabilidad de sobreajuste.
- **Normalización de Píxeles:** Los valores de los píxeles se normalizan de un rango de 0-255 a 0-1. Esto es una práctica estándar en redes neuronales, ya que ayuda a

que el proceso de optimización sea más estable y rápido.

2. Selección del Modelo (ResNet50 con Transfer Learning)

Para el modelo de clasificación de emociones, se eligió la arquitectura **ResNet50** utilizando la técnica de **Transfer Learning**. Las razones clave para esta elección son:

- **Rendimiento Comprobado:** ResNet50 es una arquitectura de red neuronal convolucional (CNN) profunda que ha demostrado un rendimiento excepcional en una amplia variedad de tareas de visión por computadora, especialmente en clasificación de imágenes. Fue pre-entrenada en el vasto dataset ImageNet, lo que significa que ya ha aprendido a extraer características visuales de alto nivel (bordes, texturas, patrones) que son transferibles a nuevas tareas.
- **Transfer Learning:** Al cargar los pesos pre-entrenados de ResNet50 y **congelar sus capas convolucionales**, se aprovecha el conocimiento adquirido en ImageNet. Esto permite al modelo comenzar con una base de características ya muy potente. Solo se entrenan las capas finales de clasificación (la "cabeza" del modelo), que se añaden encima de ResNet50. Esto es beneficioso porque:
    - **Requiere menos datos de entrenamiento:** No se necesita un dataset tan masivo como el de

ImageNet para lograr buenos resultados, lo cual es ventajoso para datasets específicos como el de emociones.

- **Entrenamiento más rápido:** Al congelar la mayor parte del modelo, menos parámetros necesitan ser actualizados durante el entrenamiento, lo que acelera el proceso.

- **Capas Adicionales:** La inclusión de una capa GlobalAveragePooling2D reduce las dimensiones del mapa de características, haciendo el modelo más eficiente. La capa Dropout (con un 20% de neuronas desactivadas aleatoriamente) es una técnica de regularización fundamental para prevenir el sobreajuste al forzar al modelo a aprender características más robustas y no depender de neuronas específicas. Finalmente, la activación Softmax en la capa de salida es apropiada para problemas de clasificación multiclase, ya que produce probabilidades para cada una de las 5 emociones.

3. Proceso de Entrenamiento

El entrenamiento del modelo se configuró con parámetros específicos y un flujo de trabajo definido:

- **División del Dataset:** El conjunto de datos pre-procesado se divide explícitamente en tres partes: 70%

para entrenamiento, 15% para validación y 15% para pruebas. Esta división tripartita es crucial:

  - **Entrenamiento (70%):** Utilizado para que el modelo aprenda los patrones de los datos y ajuste sus pesos.
  - **Validación (15%):** Utilizado durante el entrenamiento para monitorear el rendimiento del modelo en datos no vistos y ajustar hiperparámetros o aplicar técnicas como *early stopping* para prevenir el sobreajuste.
  - **Prueba (15%):** Un conjunto completamente independiente que el modelo **nunca** ve durante el entrenamiento o la validación, utilizado exclusivamente para la evaluación final imparcial del rendimiento del modelo.

- **Estratificación en la División:** Se utiliza la estrategia de estratificación (stratify) durante la división. Esto asegura que la proporción de cada clase (emoción) sea aproximadamente la misma en los conjuntos de entrenamiento, validación y prueba, lo cual es vital para evitar sesgos en el entrenamiento y obtener una evaluación más representativa, especialmente en datasets con desbalance de clases.
- **Optimizador Adam y Función de Pérdida:** Se utiliza el optimizador Adam (con una tasa de aprendizaje

LEARNING_RATE = 0.0001) y la función de pérdida CategoricalCrossentropy.

- **Adam:** Es un optimizador adaptativo muy popular y eficiente que ajusta automáticamente las tasas de aprendizaje para cada parámetro del modelo, lo que generalmente conduce a una convergencia más rápida y estable.
- **CategoricalCrossentropy:** Es la función de pérdida estándar para problemas de clasificación multiclase donde las etiquetas están en formato "one-hot encoding" (como las proporcionadas por image_dataset_from_directory con label_mode='categorical').

- **Métrica de Precisión (Accuracy):** Se monitorea la precisión (accuracy) durante el entrenamiento, que es una métrica intuitiva y directa para entender el porcentaje de clasificaciones correctas.

4. Técnicas de Evaluación

La evaluación del modelo se realiza utilizando un conjunto de técnicas para obtener una comprensión completa de su rendimiento:

- **Métricas Globales (Pérdida y Precisión):** Se calcula la pérdida y la precisión del modelo en el conjunto de prueba/validación. Estos valores proporcionan una visión general rápida del rendimiento del modelo. La

precisión indica el porcentaje total de predicciones correctas, mientras que la pérdida cuantifica el error del modelo.

- **Matriz de Confusión:** La matriz de confusión es una herramienta visual y tabular indispensable. Permite identificar:
    - **Aciertos por Clase:** Cuántas imágenes de cada emoción fueron clasificadas correctamente.
    - **Confusiones Específicas:** Cuáles clases son frecuentemente confundidas entre sí por el modelo (ej. si "triste" se confunde a menudo con "angustia"), proporcionando información valiosa para la mejora del modelo o del dataset.
- **Reporte de Clasificación (Precision, Recall, F1-Score):** Complementando la matriz de confusión, el reporte de clasificación ofrece métricas detalladas por cada clase:
    - **Precisión (Precision):** Indica la fiabilidad de las predicciones positivas del modelo para cada clase (de lo que el modelo dijo que era X, ¿cuánto fue realmente X?).
    - **Recall (Sensibilidad):** Mide la capacidad del modelo para encontrar todas las instancias positivas de cada clase (de todos los X reales, ¿cuántos encontró el modelo?).

- **F1-Score:** Es la media armónica de precisión y recall, proporcionando un balance entre ambas métricas y siendo útil para evaluar el rendimiento en clases desbalanceadas.
- Estas métricas son cruciales para identificar clases problemáticas y entender dónde el modelo necesita mejorar.

- **Gráficas de Historial de Entrenamiento:** Se generan gráficas de precisión y pérdida para los conjuntos de entrenamiento y validación a lo largo de las épocas. Estas gráficas son vitales para diagnosticar problemas como:
  - **Sobreajuste (Overfitting):** Cuando la precisión de entrenamiento sigue mejorando, pero la de validación se estanca o disminuye, o cuando la pérdida de entrenamiento baja, pero la de validación sube.
  - **Subajuste (Underfitting):** Cuando ambas métricas (precisión y pérdida) no mejoran lo suficiente, indicando que el modelo no ha aprendido adecuadamente de los datos.

En conjunto, estas decisiones y técnicas proporcionan un marco robusto y analítico para desarrollar, entrenar y evaluar un modelo de reconocimiento de emociones faciales, permitiendo una iteración y mejora continua del sistema.