DOCUMENTACIÓN - MIA - RETO 2

Ane
Imanol
Markel
Julen

INDICE

EXTRAER FRAMES DE MUESTRA DE LOS VIDEOS EN NVIDIA JETSON NANO

- Instalamos python y comprobamos la versión.

```
grupo4@grupo4-desktop:~$ python3 --version
Python 3.6.9
grupo4@grupo4-desktop:~$ pip --version
pip 21.3.1 from /home/grupo4/.local/lib/python3.6/site-packages/pip (python 3.6)
grupo4@grupo4-desktop:~$
```

- Instalamos la biblioteca OpenCV necesaria para trabajar con imágenes y videos.

```
grupo4@grupo4-desktop:~$ sudo apt-get install python3-opencv
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
```

- Instalamos el editor de texto nano.

```
grupo4@grupo4-desktop:~$ nano --version
 GNU nano, version 2.9.3
 (C) 1999-2011, 2013-2018 Free Software Foundation, Inc.
 (C) 2014-2018 the contributors to nano
 Email: nano@nano-editor.org    Web: https://nano-editor.org/
 Compiled options: --disable-libmagic --disable-wrapping-as-root --enable-utf8
grupo4@grupo4-desktop:~$
```

- Creamos el archivo nano ***config.json***

```
grupo4@grupo4-desktop:~$ nano config.json
```

- Dentro del archivo ***config.json*** con el siguiente código:

```
  GNU nano 2.9.3                        config.json

{
        "output_folder": "frames_extraidos",
        "frame_rate": 5
}
```

Explicación: este código hace que de todos los frames totales que se extraen, solo se guarde 1 cada 5 y que se guarden en la carpeta que seleccionamos, en este caso la carpeta se llama Frames Extraídos.

- Creamos el archivo nano ***extract_frames.py***

```
grupo4@grupo4-desktop:~$ nano extract_frames.py
```

- Dentro del archivo ***extract_frames.py*** con el siguiente código:

```python
import cv2
import os
import json
import sys

# Cargar la configuración desde el archivo JSON
def load_config(config_path):
    with open(config_path, 'r') as file:
        return json.load(file)
```

```python
# Función para extraer frames
def extract_frames(video_path, output_folder, frame_rate=1):
    # Obtener el nombre del video sin la ruta ni la extensión
    video_name = os.path.splitext(os.path.basename(video_path))[0]

    # Crear una subcarpeta para este video dentro de la carpeta de salida
    video_output_folder = os.path.join(output_folder, video_name)
    if not os.path.exists(video_output_folder):
        os.makedirs(video_output_folder)

    # Capturar el video
    video = cv2.VideoCapture(video_path)

    count = 0
    success = True

    while success:
        # Leer un frame del video
        success, image = video.read()
        if count % frame_rate == 0 and success:
            # Guardar el frame como imagen en la subcarpeta
            frame_filename = os.path.join(video_output_folder,
f'frame_{count}.jpg')
            cv2.imwrite(frame_filename, image)
        count += 1

    # Liberar el objeto de captura
    video.release()
    print(f"Frames extraídos de {video_name}: {count}")

# Uso de la función
if __name__ == "__main__":
    config = load_config('config.json')  # Cargar la configuración

    # Comprobar si se pasó una ruta de video como argumento
    if len(sys.argv) > 1:
        video_path = sys.argv[1]  # Toma la ruta del video del primer
argumento
    else:
        raise ValueError("Por favor, proporciona una ruta al video como
argumento.")
```

```
    # Extraer los demás parámetros de config.json
    output_folder = config['output_folder']
    frame_rate = config['frame_rate']

    # Llamar a la función para extraer frames
    extract_frames(video_path, output_folder, frame_rate)
```

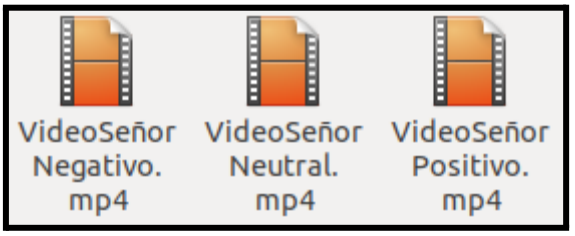
Explicación: Este código extrae todos los frames del video que le pedimos al ejecutar el script y guarda la cantidad de frames que le hemos dicho en el anterior archivo que guarde (en este caso, 1 cada 5), y los frames se guardan en una carpeta con el nombre del propio vídeo y si no existe, se crea la carpeta de forma automática.

- Una vez hecho todo esto, solo tendremos que ejecutar el script, poniendo la ruta del video que deseemos:

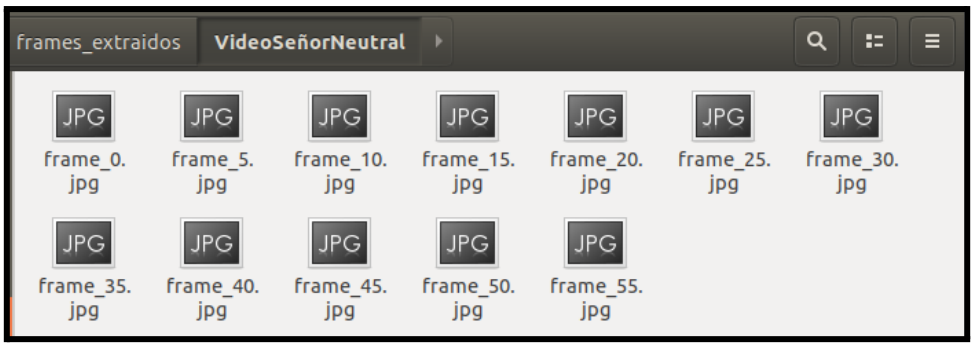
python3 extract_frames.py /ruta/video.mp4

En nuestro caso la ruta y nombres de los videos son los siguientes:

```
grupo4@grupo4-desktop:~$ python3 extract_frames.py /home/grupo4/Desktop/VIDEOS/V
ideoSeñorNegativo.mp4
Frames extraídos de VideoSeñorNegativo: 61
grupo4@grupo4-desktop:~$ python3 extract_frames.py /home/grupo4/Desktop/VIDEOS/V
ideoSeñorPositivo.mp4
Frames extraídos de VideoSeñorPositivo: 61
grupo4@grupo4-desktop:~$ python3 extract_frames.py /home/grupo4/Desktop/VIDEOS/V
ideoSeñorNeutral.mp4
Frames extraídos de VideoSeñorNeutral: 61
grupo4@grupo4-desktop:~$
```

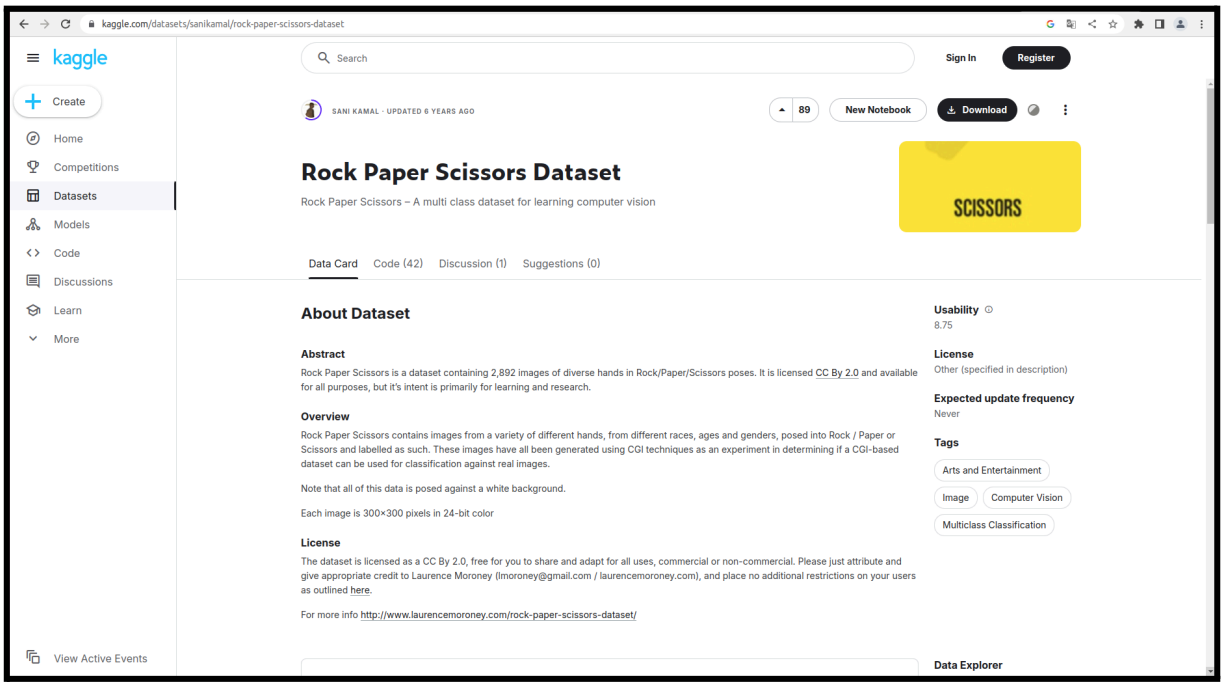


Esta sería una de las 3 carpetas creadas con los frames de uno de los videos anteriormente mencionados.



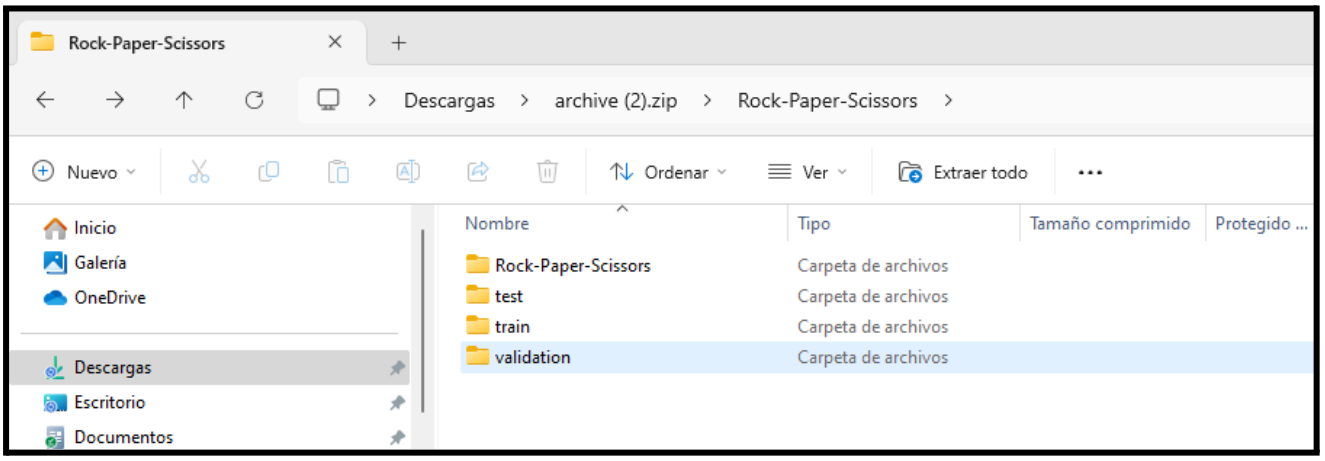
RED NEURONAL

Para nuestra red neuronal, hemos usado imágenes de un dataset de Piedra, Papel o Tijera:

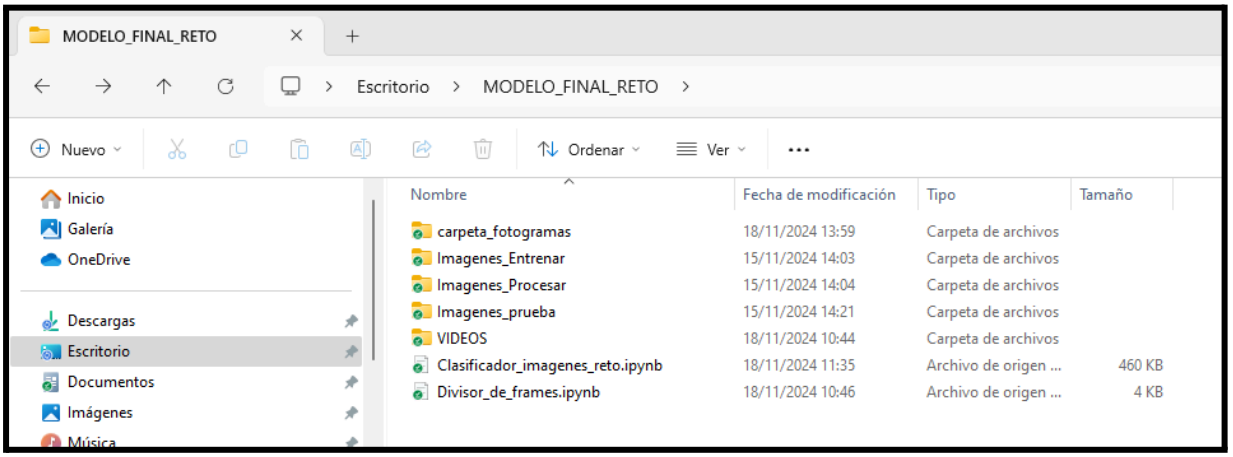


enlace para descargar el dataset:
https://www.kaggle.com/datasets/sanikamal/rock-paper-scissors-dataset

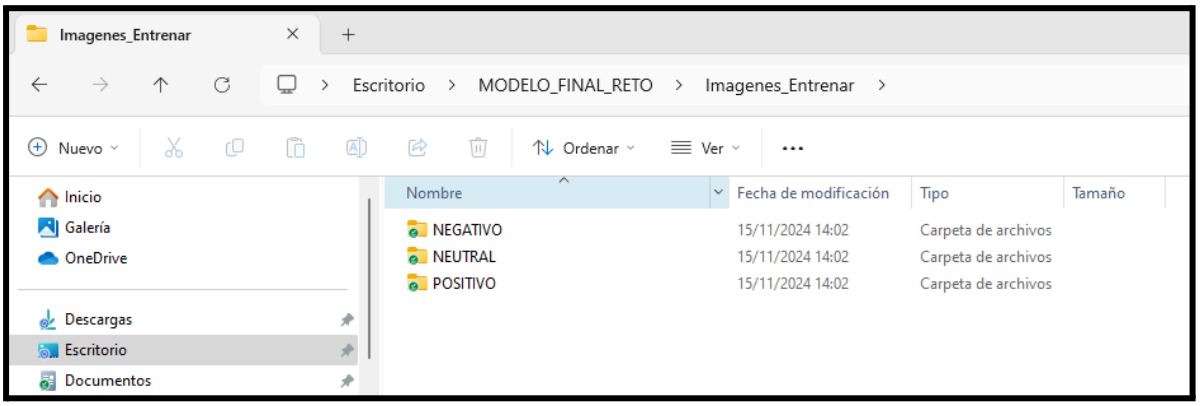
- Esta sería la carpeta que se nos descarga, del dataset de kaggle:



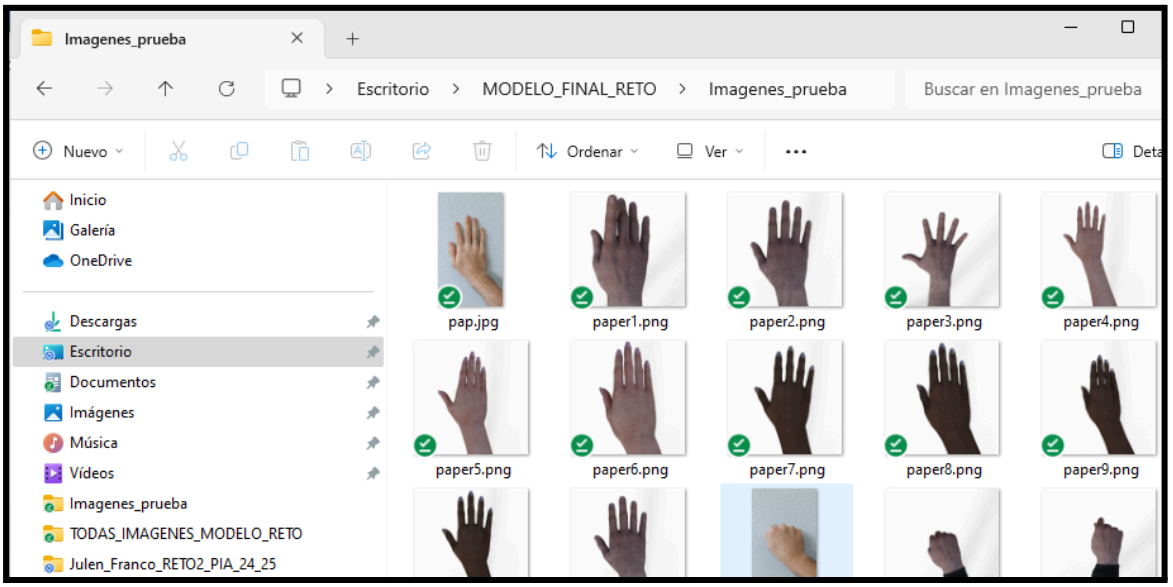
Esta sería la carpeta que tenemos nosotros, con el siguiente contenido:
- **carpeta_fotogramas:** en esta carpeta se guardan los fotogramas sacados de los videos.
- **Imagenes_Entrenar:** en esta carpeta están las imagenes que usamos para entrenar el modelo.
- Imagenes_Procesar: en esta carpeta están las imagenes que usamos que el modelo testee.
- Imagenes_prueba: en esta carpeta están las imagenes que usamos para para hacer pruebas con nuestro modelo.
- VIDEOS: en esta carpeta guardamos los videos para extraer frames de ellos.
- Clasificador_imagenes_reto: en este archivo tenemos nuestro modelo.
- Divisor_de_frames: en este archivo tenemos el código para extraer frames de los videos.



Dentro de cada carpeta de Imagentes_entrenamiento y Imagenes_prueba, separaremos las imágenes en 3 carpetas: Negativo, Positivo y Neutral



Aqui vemos una pequeña muestra de las imágenes que contienen las carpetas, en este caso la carpeta: **Imagenes_prueba**



en el caso de nuestro modelo, hemos decidido hacerlo con imágenes de piedra papel o tijera

- Piedra = POSITIVO
- Papel = NEUTRAL
- Tijera = NEGATIVO

FALTA:
capturas modelo
explicar capturas modelo

capturas de alguna prueba (buena)

EXPLICACIÓN DEL CÓDIGO DE LA RED

-Aquí importamos las librerías necesarias para ejecutar nuestro código.

```
# Importamos librerías
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import os
```

-Aquí colocamos las rutas de nuestras carpetas de entrenamiento y procesamiento.

```
# Ruta al conjunto de datos descomprimido
ruta_entrenamiento = 'Imagenes_Entrenar'
ruta_pruebas = 'Imagenes_Procesar'
```

-Este bloque es para obtener las clases que contienen las carpetas.

```
# Obtener los nombres de las carpetas (clases)
class_names = os.listdir(ruta_entrenamiento)
class_names_pr = os.listdir(ruta_pruebas)
```

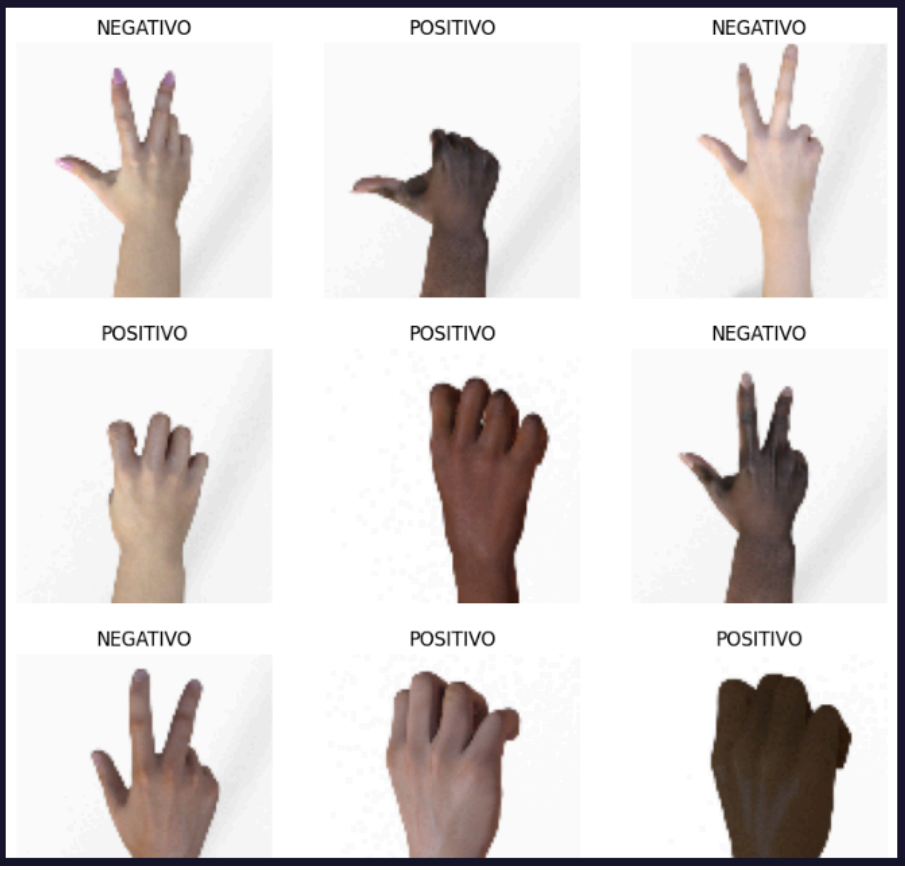
-Aquí creamos los conjuntos de datos:
ajustamos las imagenes a tamaño 100 x 100

```
# Crear los conjuntos de datos
datos_entrenamiento = tf.keras.preprocessing.image_dataset_from_directory(
    ruta_entrenamiento,
    image_size=(100, 100),
    batch_size=32,
    class_names=class_names,  # Usar la lista de nombres de clases
    seed=123  # Semilla para reproducibilidad
)

datos_pruebas = tf.keras.preprocessing.image_dataset_from_directory(
    ruta_pruebas,
    image_size=(100, 100),
    batch_size=32,
    class_names=class_names_pr,  # Usar la lista de nombres de clases
    seed=123  # Semilla para reproducibilidad
)
```

-Con este bloque de código, podemos visualizar algunas de las imágenes de entrenamiento.

```python
def mostrar_imagenes(dataset, class_names, num_imagenes=9):
    plt.figure(figsize=(10, 10))
    for images, labels in dataset.take(1):
        for i in range(min(num_imagenes, images.shape[0])):  # Asegurarse de no exceder el número de imágenes
            ax = plt.subplot(3, 3, i + 1)
            plt.imshow(images[i].numpy().astype("uint8"))
            plt.title(class_names[labels[i].numpy()])  # Asegúrate de convertir etiquetas a numpy
            plt.axis("off")
    plt.show()

# Llamar a la función para mostrar imágenes
mostrar_imagenes(datos_entrenamiento, class_names)
```

[5]



-Con este bloque de código, normalizamos las imágenes

```python
# Normalizar imágenes (escala de 0 a 1)
normalizar = layers.Rescaling(1./255)
datos_entrenamiento = datos_entrenamiento.map(lambda x, y: (normalizar(x), y))
datos_pruebas = datos_pruebas.map(lambda x, y: (normalizar(x), y))
```

-Aquí creamos nuestro modelo

- Numerosas capas con distintas cantidades de neuronas para garantizar un entrenamiento correcto
- Dropout del 50% para evitar el sobreajuste

```
from tensorflow.keras import regularizers

modelo = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', kernel_regularizer=regularizers.l2(0.01), input_shape=(100, 100, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.BatchNormalization(),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.Flatten(),
    # Dropout para evitar sobreajuste
    layers.Dropout(0.5),
    layers.Dense(64, activation='relu'),
    layers.Dense(len(class_names), activation='softmax')
])
```

-Compilamos el modelo

```
# Compilar el modelo
modelo.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

-Aquí entrenamos nuestro modelo con 10 epocas

```
# Entrenar el modelo
historial = modelo.fit(datos_entrenamiento, epochs=10, validation_data=datos_pruebas)
```

-Aquí vemos el resultado que ha dado nuestro modelo entrenado

Podemos ver como el valor de precisión se ha mantenido muy elevado, y el valor de perdida ha ido disminuyendo.

```
Epoch 1/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 29s 343ms/step - accuracy: 0.7210 - loss: 2.4123 - val_accuracy: 0.3441 - val_loss: 1.1930
Epoch 2/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 350ms/step - accuracy: 0.9981 - loss: 0.0521 - val_accuracy: 0.7285 - val_loss: 0.7132
Epoch 3/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 27s 334ms/step - accuracy: 1.0000 - loss: 0.0369 - val_accuracy: 0.6452 - val_loss: 0.7685
Epoch 4/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 26s 333ms/step - accuracy: 1.0000 - loss: 0.0349 - val_accuracy: 0.5349 - val_loss: 1.2576
Epoch 5/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 358ms/step - accuracy: 1.0000 - loss: 0.0332 - val_accuracy: 0.5349 - val_loss: 1.5970
Epoch 6/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 351ms/step - accuracy: 1.0000 - loss: 0.0317 - val_accuracy: 0.6586 - val_loss: 1.1812
Epoch 7/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 27s 339ms/step - accuracy: 1.0000 - loss: 0.0303 - val_accuracy: 0.7608 - val_loss: 0.9192
Epoch 8/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 27s 335ms/step - accuracy: 1.0000 - loss: 0.0289 - val_accuracy: 0.8145 - val_loss: 0.8633
Epoch 9/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 27s 345ms/step - accuracy: 1.0000 - loss: 0.0276 - val_accuracy: 0.8145 - val_loss: 0.9408
Epoch 10/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 28s 348ms/step - accuracy: 1.0000 - loss: 0.0263 - val_accuracy: 0.7903 - val_loss: 1.0202
```

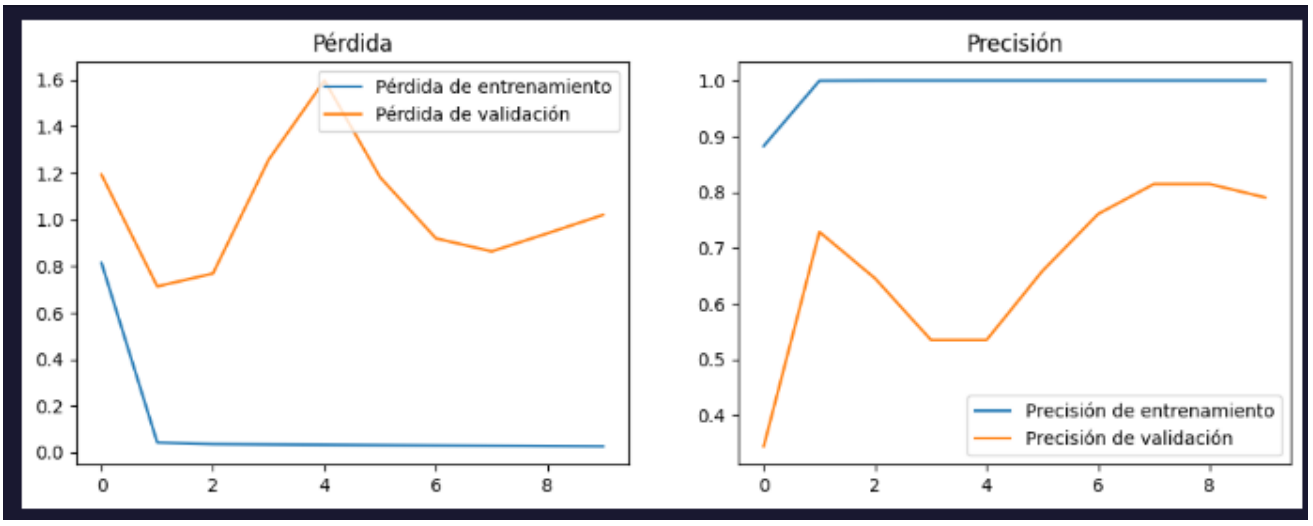
-Aquí vemos el valor de precisión de nuestro modelo y el valor de pérdida, en este caso:
Accuracy (precisión): 0.76
Pérdida: 1.09

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 54ms/step - accuracy: 0.7597 - loss: 1.0920
Pérdida en prueba: 1.0202, Precisión en prueba: 0.7903
```

-Con este bloque, visualizamos una gráfica de los valores de precisión y de pérdida durante el entrenamiento y el testeo:

```python
# Visualización de la pérdida y precisión durante el entrenamiento
plt.figure(figsize=(12, 4))

# Pérdida
plt.subplot(1, 2, 1)
plt.plot(historial.history['loss'], label='Pérdida de entrenamiento')
plt.plot(historial.history['val_loss'], label='Pérdida de validación')
plt.title('Pérdida')
plt.legend()

# Precisión
plt.subplot(1, 2, 2)
plt.plot(historial.history['accuracy'], label='Precisión de entrenamiento')
plt.plot(historial.history['val_accuracy'], label='Precisión de validación')
plt.title('Precisión')
plt.legend()

plt.show()
```

-En este caso:



-Y en el último bloque, tenemos los siguientes fragmentos de código:

-Importamos las librerías necesarias

```
import tensorflow as tf
from tensorflow.keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt
import requests
from io import BytesIO
from PIL import Image  # Importar Image desde PIL
```

-Podemos seleccionar si hacer una prueba con una imagen o de Google o una imagen guardada en nuestro ordenador, gracias al siguiente código:

```
# EL CONTENIDO ENTRE LAS DOS BARRAS DE ABAJO SE PUEDE CAMBIAR, DEPENDIENDO SI QUEREMOS USAR UNA IMAGEN DE GOOGLE O DE ARCHIVO
# ----------------------------------------SI QUEREMOS USAR UNA IMAGEN DE GOOGLE, USAREMOS ESTE FRAGMENTO DE CODIGO ↓↓↓↓ ------------
# Cargar la imagen
ruta_imagen = input("Ingrese la ruta de la imagen: ")
imagen_prueba = Image.open(ruta_imagen)
# ----------------------------------------SI QUEREMOS USAR UNA IMAGEN DE GOOGLE, USAREMOS ESTE FRAGMENTO DE CODIGO ↓↓↓↓ ------------
'''
# URL de la imagen que quieres probar
url_imagen = input('')


# Descargar la imagen desde la URL
response = requests.get(url_imagen)
imagen_prueba = Image.open(BytesIO(response.content))
'''
# ---------------------------------------------------------------------------------------------------------------------------------
```

-Con este código ajustamos la imagen de prueba y la mostramos

```
# Convertir a RGB si es necesario
if imagen_prueba.mode != 'RGB':
    imagen_prueba = imagen_prueba.convert('RGB')

# Ajustar tamaño a 100x100
imagen_prueba = imagen_prueba.resize((100, 100))

# Mostrar la imagen descargada
plt.imshow(imagen_prueba)
plt.axis('off')
plt.show()
```

-Procesamos la imagen y hacemos la predicción:

```python
# Convertir la imagen en array y preprocesar
imagen_array = image.img_to_array(imagen_prueba)  # Convertir a array de numpy

# Añadir dimensión para el lote (batch)
imagen_array = np.expand_dims(imagen_array, axis=0)

# Normalizar a rango 0-1
imagen_array = imagen_array / 255.0

# Hacer la predicción
prediccion = modelo.predict(imagen_array)

# Mostrar los resultados
indice_prediccion = np.argmax(prediccion[0])  # Índice de la clase con mayor probabilidad
nombre_clase = class_names[indice_prediccion]  # Obtener el nombre de la clase
confianza = 100 * np.max(prediccion[0])  # Confianza en porcentaje

print(f"Predicción: {nombre_clase} con una confianza de {confianza:.2f}%")
```

PRUEBAS DEL MODELO

PRUEBA 1

Aquí hacemos una prueba con el gesto de **roca**, que debería ser positivo, y como podemos ver, lo predice correctamente.

PRUEBA 2

Aquí hacemos una prueba con el gesto de **tijera**, que debería ser negativo, y como podemos ver, lo predice correctamente.

PRUEBA 3

Aquí hacemos una prueba con el gesto de **papel**, que debería ser neutral, y como podemos ver, lo predice correctamente.