MANUAL DE USUARIO

INTRODUCCIÓN

Este manual detalla el uso de varias clases y modelos de clasificación de imágenes implementados en el código. Se incluyen descripciones específicas de cada modelo y pasos para cargar datos, seleccionar un modelo, entrenar, evaluar y ajustar hiperparámetros.

Requisitos Previos

Hardware

- Procesador: Intel Core i5 o superior.
- Memoria RAM: 8 GB mínimo.
- Almacenamiento: 10 GB libres.

Software

- Sistema Operativo: Windows 10 / Ubuntu 20.04 o superior.
- Dependencias:
    - Python 3.9 o superior.
    - Librerías: TensorFlow, OpenCV, Matplotlib, Seaborn.
    - Plataforma gráfica: Tkinter.

Conjunto de datos de imágenes organizadas en carpetas en Google Drive.

- **Carpeta principal**: **data**
- **Subcarpeta 1**: **purpura_reumatoidea**
- **Subcarpeta 2**: **saludable**

Paso 1: Configuración del Entorno

1.1: Importar las Bibliotecas Necesarias

Primero, importaremos todas las bibliotecas necesarias para este proyecto:

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
from google.colab import drive
```

1.2: Montar Google Drive

Montamos Google Drive para acceder a los datos almacenados en él:

```
drive.mount('/content/drive')
```

Paso 2: Preparar los Datos

2.1: Definir el Directorio de Datos

Definimos la ubicación de nuestro conjunto de datos en Google Drive:

```
data_dir = '/content/drive/MyDrive/data'
```

2.2: Crear Estructura de Carpetas

Creamos las carpetas necesarias para almacenar las imágenes (si no existen):

```
os.makedirs(data_dir, exist_ok=True)
os.makedirs(os.path.join(data_dir, 'purpura_reumatoidea'), exist_ok=True)
os.makedirs(os.path.join(data_dir, 'saludable'), exist_ok=True)
```

2.3: Cargar y Preprocesar Imágenes

Definimos una función para cargar y preprocesar las imágenes desde las carpetas:

```
def load_data(data_dir, categories, img_size):
    data = []
    for category in categories:
        path = os.path.join(data_dir, category)
        class_num = categories.index(category)
        for img in os.listdir(path):
            try:
                img_array = plt.imread(os.path.join(path, img))
                resized_img = tf.image.resize(img_array, (img_size, img_size))
                data.append([resized_img, class_num])
            except Exception as e:
                pass
    return data
```

2.4: Definir Categorías y Cargar los Datos

Definimos las categorías de las imágenes y cargamos los datos:

```
categories = ['purpura_reumatoidea', 'saludable']
img_size = 128
data = load_data(data_dir, categories, img_size)
```

2.5: Preparar los Datos para el Modelo

Mezclamos los datos y los dividimos en características (X) y etiquetas (y):

```
np.random.shuffle(data)
X = []
y = []

for features, label in data:
    X.append(features)
    y.append(label)

X = np.array(X) / 255.0  # Normalizar las imágenes
y = np.array(y)
```

2.6: Dividir los Datos en Entrenamiento y Prueba

Dividimos los datos en conjuntos de entrenamiento y prueba:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Paso 3: Construcción del Modelo

3.1: Generación de Datos Aumentados

Definimos un generador de datos para realizar aumentos en las imágenes de entrenamiento:

```
train_datagen = ImageDataGenerator(
    rotation_range=20,
    zoom_range=0.15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    horizontal_flip=True,
    fill_mode="nearest"
)

train_generator = train_datagen.flow(X_train, y_train, batch_size=32)
```

3.2: Definir la Arquitectura del Modelo

Construimos la red neuronal convolucional usando la API secuencial de Keras:

```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(img_size, img_size, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(2, activation='softmax')
])
```

3.3: Compilar el Modelo

Compilamos el modelo con el optimizador Adam y la función de pérdida **sparse_categorical_crossentropy**:

```
model.compile(optimizer=Adam(lr=0.001),
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Paso 4: Entrenamiento del Modelo

Entrenamos el modelo con los datos de entrenamiento aumentados y validamos con los datos de prueba:

```
history = model.fit(train_generator, epochs=25, validation_data=(X_test, y_test))
```

Paso 5: Evaluación del Modelo

Evaluamos el modelo en el conjunto de prueba y mostramos la pérdida y precisión:

```
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Pérdida: {loss}, Precisión: {accuracy}')
```

Paso 6: Visualización de Resultados

6.1: Graficar Precisión y Pérdida

Grafica la precisión y la pérdida del modelo tanto para el conjunto de entrenamiento como para el de validación a lo largo de las épocas:

```
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.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.show()

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.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

6.2: Matriz de Confusión

Genera predicciones para el conjunto de prueba, calcula la matriz de confusión y la gráfica:

```
y_pred_prob = model.predict(X_test) y_pred
= np.argmax(y_pred_prob, axis=1)

cm = confusion_matrix(y_test, y_pred) plt.figure(figsize=(8,
6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=categories, yticklabels=categories) plt.xlabel('Predicted')
plt.ylabel('Actual') plt.title('Confusion Matrix') plt.show()
```

6.3: Reporte de Clasificación

Muestra un reporte de clasificación que incluye precisión, recall y F1-score para cada clase:

```
report = classification_report(y_test, y_pred, target_names=categories)
print('Classification Report:') print(report)
```

6.4: Curva ROC y AUC

Calcula y grafica la curva ROC y el AUC para evaluar la capacidad de discriminación del modelo:

```
fpr, tpr, _ = roc_curve(y_test, y_pred_prob[:, 1]) roc_auc
= auc(fpr, tpr)
 plt.figure() plt.plot(fpr, tpr, color='darkorange', lw=2,
label='ROC curve (area =
%0.2f)' % roc_auc) plt.plot([0, 1], [0, 1], color='navy',
lw=2, linestyle='--') plt.xlim([0.0, 1.0]) plt.ylim([0.0,
1.05]) plt.xlabel('False Positive Rate') plt.ylabel('True
Positive Rate') plt.title('Receiver Operating
Characteristic (ROC)') plt.legend(loc="lower right")
plt.show()
```

7 instrucciones de Uso

7.1 Inicio del Sistema

1. Ejecute el archivo principal: python main.py
2. Aparecerá la interfaz gráfica principal.

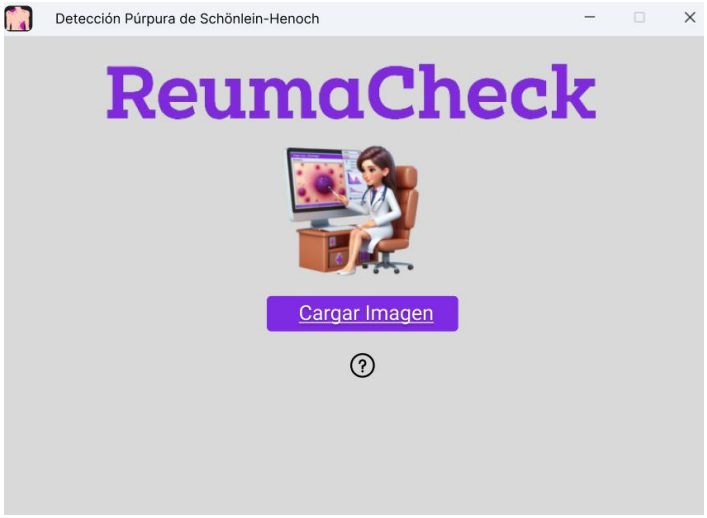

Ilustración 1 Interfaz Principal

7.2 Carga de imágenes

- Pulse el botón **"Cargar Imagen"**.
- Seleccione una imagen de alta calidad que muestre la piel del paciente.

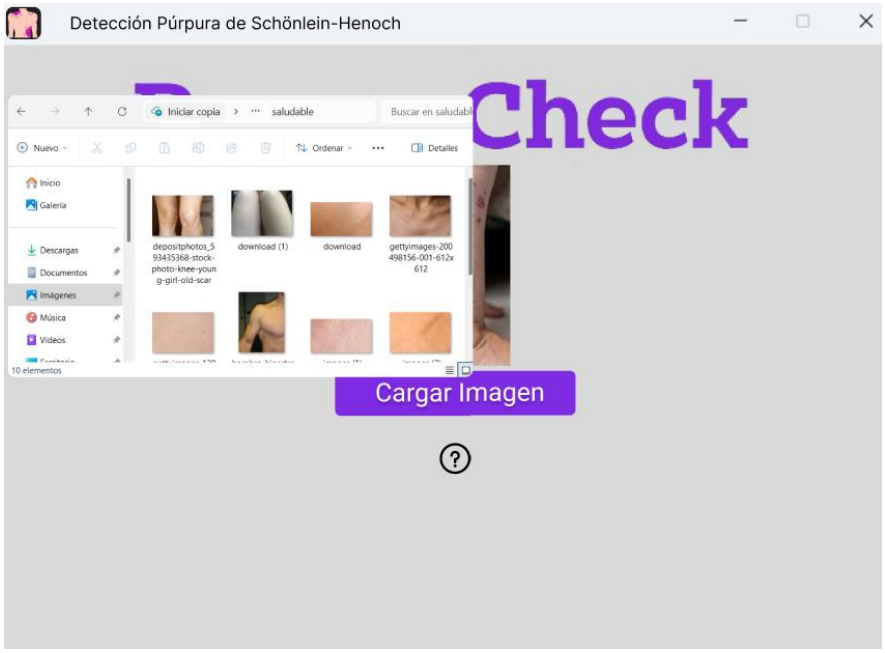

Ilustración 2 Adjuntar imagen

7.3 Análisis de la imagen

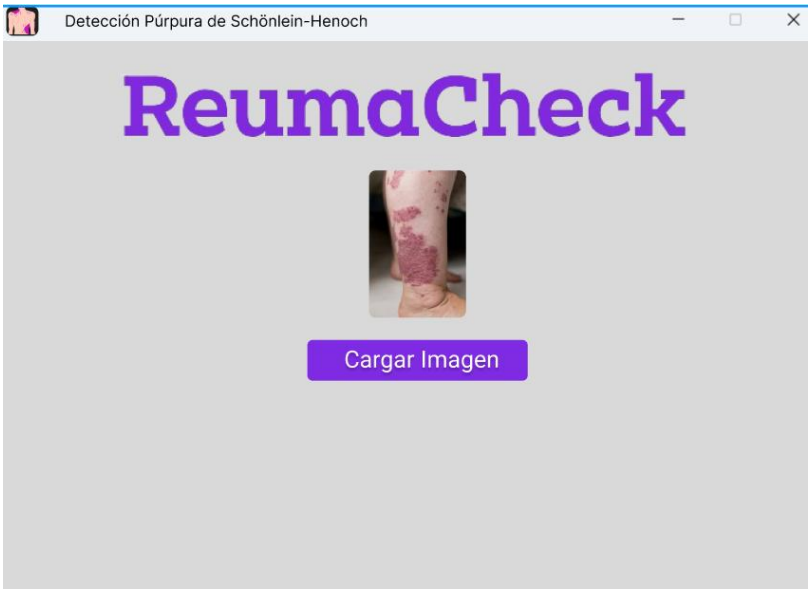

Ilustración 3 Imagen tomada

- Espere unos segundos mientras el sistema procesa la información.

7.4 Visualización de resultados

- Los resultados aparecerán en pantalla:
    - **Probabilidad de PR**: Un porcentaje que indica la posibilidad de PR.