!pip install streamlit tensorflow keras opencv-python matplotlib seaborn pandas numpy scikit-learn plotly keras-tuner pyngrok fpdf

from google.colab import drive

drive.mount('/content/drive')

import os

import shutil

from tqdm import tqdm

def copy\_dataset(input\_dir, output\_dir):

    os.makedirs(output\_dir, exist\_ok=True)

    subsets = ['train', 'valid', 'test']

    classes = ['Healthy', 'Disease']

    for subset in subsets:

        for label in classes:

            input\_class\_dir = os.path.join(input\_dir, subset, label)

            output\_class\_dir = os.path.join(output\_dir, subset, label)

            os.makedirs(output\_class\_dir, exist\_ok=True)

            print(f"Copiando clase '{label}' del conjunto '{subset}'")

            for file\_name in tqdm(os.listdir(input\_class\_dir)):

                input\_path = os.path.join(input\_class\_dir, file\_name)

                output\_path = os.path.join(output\_class\_dir, file\_name)

                try:

                    shutil.copy2(input\_path, output\_path)

                except Exception as e:

                    print(f"Error copiando {file\_name}: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

    base\_input\_dir = "/content/drive/MyDrive/Avocado Augmneted\_Dataset"

    base\_output\_dir = "/content/avocado\_dataset"

    copy\_dataset(base\_input\_dir, base\_output\_dir)

%%writefile model\_utils.py

import tensorflow as tf

from tensorflow.keras.models import Model

from tensorflow.keras.layers import (

    Input, Conv2D, MaxPooling2D, GlobalAveragePooling2D,

    Dense, Dropout, BatchNormalization

)

from tensorflow.keras.optimizers import Adam

import numpy as np

from tensorflow.keras.applications import DenseNet121, EfficientNetB0

def load\_models():

    models = {}

    # CNN 2D Personalizada para RGB

    input\_shape = (224, 224, 3)

    inputs = Input(input\_shape)

    x = Conv2D(32, (3,3), activation='relu')(inputs)

    x = MaxPooling2D((2,2))(x)

    x = BatchNormalization()(x)

    x = Conv2D(64, (3,3), activation='relu')(x)

    x = MaxPooling2D((2,2))(x)

    x = BatchNormalization()(x)

    x = Conv2D(128, (3,3), activation='relu')(x)

    x = MaxPooling2D((2,2))(x)

    x = BatchNormalization()(x)

    x = GlobalAveragePooling2D()(x)

    x = Dense(256, activation='relu')(x)

    x = Dropout(0.5)(x)

    outputs = Dense(1, activation='sigmoid')(x)

    model = Model(inputs=inputs, outputs=outputs)

    model.compile(

        optimizer=Adam(learning\_rate=0.0005),

        loss='binary\_crossentropy',

        metrics=['accuracy', tf.keras.metrics.AUC(name='auc'),

                 tf.keras.metrics.Precision(name='precision'),

                 tf.keras.metrics.Recall(name='recall')]

    )

    models['CNN 2D Personalizada'] = model

    # DenseNet121

    densenet\_input = Input(shape=(224, 224, 3))

    base\_densenet = DenseNet121(weights='imagenet', include\_top=False, input\_tensor=densenet\_input)

    base\_densenet.trainable = False

    x = base\_densenet.output

    x = GlobalAveragePooling2D()(x)

    x = Dense(256, activation='relu')(x)

    x = Dropout(0.5)(x)

    output = Dense(1, activation='sigmoid')(x)

    densenet\_model = Model(inputs=densenet\_input, outputs=output)

    densenet\_model.compile(

        optimizer=Adam(learning\_rate=0.0005),

        loss='binary\_crossentropy',

        metrics=['accuracy', tf.keras.metrics.AUC(name='auc'),

                 tf.keras.metrics.Precision(name='precision'),

                 tf.keras.metrics.Recall(name='recall')]

    )

    models["DenseNet121"] = densenet\_model

    # Modelo 3: EfficientNetB0

    efficient\_input = Input(shape=(224, 224, 3))

    base\_efficient = EfficientNetB0(weights='imagenet', include\_top=False, input\_tensor=efficient\_input)

    base\_efficient.trainable = False

    x = base\_efficient.output

    x = GlobalAveragePooling2D()(x)

    x = Dense(256, activation='relu')(x)

    x = Dropout(0.5)(x)

    output = Dense(1, activation='sigmoid')(x)

    efficient\_model = Model(inputs=efficient\_input, outputs=output)

    efficient\_model.compile(

        optimizer=Adam(learning\_rate=0.0005),

        loss='binary\_crossentropy',

        metrics=['accuracy', tf.keras.metrics.AUC(name='auc'),

                 tf.keras.metrics.Precision(name='precision'),

                 tf.keras.metrics.Recall(name='recall')]

    )

    models["EfficientNetB0"] = efficient\_model

    return models

def predict\_image(model, image):

    if image.shape[-1] != 3:

        raise ValueError("La imagen debe tener 3 canales RGB.")

    image = np.expand\_dims(image, axis=0)

    prediction = model.predict(image)

    confidence = float(prediction[0][0])

    diagnosis = 1 if confidence > 0.5 else 0

    heatmap = generate\_saliency\_map(model, image[0])

    return diagnosis, confidence, heatmap

def generate\_saliency\_map(model, image):

    if image.ndim == 3:

        image\_tensor = tf.convert\_to\_tensor(np.expand\_dims(image, axis=0), dtype=tf.float32)

    elif image.ndim == 4:

        image\_tensor = tf.convert\_to\_tensor(image, dtype=tf.float32)

    else:

        raise ValueError("Imagen debe tener 3 (HWC) o 4 (BHWC) dimensiones.")

    with tf.GradientTape() as tape:

        tape.watch(image\_tensor)

        prediction = model(image\_tensor)

    gradients = tape.gradient(prediction, image\_tensor)

    saliency\_map = tf.reduce\_max(tf.abs(gradients), axis=-1)[0].numpy()

    if saliency\_map.max() != saliency\_map.min():

        saliency\_map = (saliency\_map - saliency\_map.min()) / (saliency\_map.max() - saliency\_map.min())

    else:

        saliency\_map = np.zeros\_like(saliency\_map)

    return saliency\_map

%%writefile preprocessing.py

from skimage.io import imread

from skimage.transform import resize

from skimage.exposure import equalize\_adapthist

import numpy as np

from skimage import img\_as\_ubyte

import cv2

def preprocess\_image(image, target\_size=(224, 224)):

        image = img\_as\_ubyte(image)

        image = cv2.cvtColor(image, cv2.COLOR\_RGB2LAB)

        l, a, b = cv2.split(image)

        clahe = cv2.createCLAHE(clipLimit=2.5, tileGridSize=(8, 8))

        cl = clahe.apply(l)

        limg = cv2.merge((cl, a, b))

        image = cv2.cvtColor(limg, cv2.COLOR\_LAB2RGB)

        image = image.astype(np.float32) / 255.0

        return image

def load\_and\_preprocess\_image(image\_path, target\_size=(224, 224)):

    image = imread(image\_path)  # RGB

    image\_original = resize(image, target\_size, mode='reflect', anti\_aliasing=True)

    image\_preprocessed = preprocess\_image(image, target\_size)

    return image\_preprocessed, image\_original

#%%writefile train\_avocado.py

import tensorflow as tf

import os

import numpy as np

import pandas as pd

from datetime import datetime

from sklearn.utils.class\_weight import compute\_class\_weight

from collections import Counter

from model\_utils import load\_models

from skimage.io import imread

from skimage.transform import resize

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, TensorBoard

from skimage import img\_as\_ubyte

import cv2

# Cargar dataset

def load\_dataset\_avocado(data\_dir):

    def get\_data(subset):

        subset\_dir = os.path.join(data\_dir, subset)

        disease\_cases = [os.path.join(subset\_dir, 'Disease', f) for f in os.listdir(os.path.join(subset\_dir, 'Disease'))]

        healthy\_cases = [os.path.join(subset\_dir, 'Healthy', f) for f in os.listdir(os.path.join(subset\_dir, 'Healthy'))]

        disease\_labels = [1] \* len(disease\_cases)

        healthy\_labels = [0] \* len(healthy\_cases)

        cases = disease\_cases + healthy\_cases

        labels = disease\_labels + healthy\_labels

        return cases, labels

    train\_cases, train\_labels = get\_data('train')

    val\_cases, val\_labels = get\_data('valid')

    test\_cases, test\_labels = get\_data('test')

    return (train\_cases, train\_labels), (val\_cases, val\_labels), (test\_cases, test\_labels)

def mostrar\_distribucion\_clases(train\_labels, val\_labels):

    print("\nDistribución de clases:")

    print(f" - Train: {Counter(train\_labels)}")

    print(f" - Valid: {Counter(val\_labels)}")

# Generador de datos

class AvocadoImageGenerator(tf.keras.utils.Sequence):

    def \_\_init\_\_(self, cases, labels, batch\_size=32, image\_size=(224, 224), augment=False, \*\*kwargs):

        super().\_\_init\_\_(\*\*kwargs)

        self.cases = cases

        self.labels = labels

        self.batch\_size = batch\_size

        self.image\_size = image\_size

        self.augment = augment

        self.indices = np.arange(len(self.cases))

        self.augmenter = ImageDataGenerator(

            rotation\_range=10,

            width\_shift\_range=0.05,

            height\_shift\_range=0.05,

            zoom\_range=0.1,

            horizontal\_flip=True,

            fill\_mode='reflect'

        ) if augment else None

    def preprocess\_image(self, image):

        # Preprocesamiento LAB + CLAHE

        image = img\_as\_ubyte(image)  # Convierte de float a uint8 si fuera necesario

        image = cv2.cvtColor(image, cv2.COLOR\_RGB2LAB)

        l, a, b = cv2.split(image)

        clahe = cv2.createCLAHE(clipLimit=2.5, tileGridSize=(8, 8))

        cl = clahe.apply(l)

        limg = cv2.merge((cl, a, b))

        image = cv2.cvtColor(limg, cv2.COLOR\_LAB2RGB)

        # Normaliza a rango [0,1]

        image = image.astype(np.float32) / 255.0

        return image

    def \_\_len\_\_(self):

        return int(np.ceil(len(self.cases) / self.batch\_size))

    def \_\_getitem\_\_(self, index):

        batch\_indices = self.indices[index \* self.batch\_size:(index + 1) \* self.batch\_size]

        batch\_cases = [self.cases[i] for i in batch\_indices]

        batch\_labels = [self.labels[i] for i in batch\_indices]

        batch\_images = []

        for path in batch\_cases:

            image = imread(path)

            image = resize(image, self.image\_size, mode='reflect', anti\_aliasing=True)

            image = self.preprocess\_image(image)

            batch\_images.append(image)

        batch\_images = np.array(batch\_images, dtype=np.float32)

        batch\_labels = np.array(batch\_labels, dtype=np.float32)

        if self.augmenter:

            aug\_iter = self.augmenter.flow(batch\_images, batch\_labels, batch\_size=self.batch\_size, shuffle=False)

            batch\_images, batch\_labels = next(aug\_iter)

        return batch\_images, batch\_labels

    def on\_epoch\_end(self):

        np.random.shuffle(self.indices)

# Entrenamiento

def train\_model(model, train\_generator, val\_generator, epochs, model\_name='avocado\_model', class\_weight=None):

    callbacks = [

        ModelCheckpoint(f'models/{model\_name}.keras', monitor='val\_auc', save\_best\_only=True, mode='max', verbose=1),

        EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True, verbose=1),

        ReduceLROnPlateau(monitor='val\_loss', factor=0.2, patience=5, min\_lr=1e-6, verbose=1),

        TensorBoard(log\_dir=f'logs/{model\_name}\_{datetime.now().strftime("%Y%m%d-%H%M%S")}')

    ]

    history = model.fit(

        train\_generator,

        validation\_data=val\_generator,

        epochs=epochs,

        callbacks=callbacks,

        class\_weight=class\_weight,

        verbose=1

    )

    pd.DataFrame(history.history).to\_csv(f'models/{model\_name}\_history.csv', index=False)

    return history

# Ejecución principal

if \_\_name\_\_ == "\_\_main\_\_":

    data\_dir = '/content/avocado\_dataset'

    os.makedirs('models', exist\_ok=True)

    os.makedirs('logs', exist\_ok=True)

    (train\_cases, train\_labels), (val\_cases, val\_labels), (test\_cases, test\_labels) = load\_dataset\_avocado(data\_dir)

    mostrar\_distribucion\_clases(train\_labels, val\_labels)

    class\_weights = compute\_class\_weight(class\_weight='balanced', classes=np.unique(train\_labels), y=train\_labels)

    class\_weight = dict(enumerate(class\_weights))

    models = load\_models()

    for modelo\_objetivo in ["CNN 2D Personalizada","EfficientNetB0","DenseNet121"]:

        print(f"\nEntrenando modelo: {modelo\_objetivo}")

        train\_gen = AvocadoImageGenerator(train\_cases, train\_labels, batch\_size=32, augment=True)

        val\_gen = AvocadoImageGenerator(val\_cases, val\_labels, batch\_size=32, augment=False)

        train\_model(

            models[modelo\_objetivo],

            train\_gen,

            val\_gen,

            epochs=50,

            model\_name=f"{modelo\_objetivo.lower().replace(' ', '\_')}\_avocado",

            class\_weight=class\_weight

        )

    for modelo\_objetivo in ["CNN 2D Personalizada","EfficientNetB0","DenseNet121"]:

        print(f"\nEntrenando modelo: {modelo\_objetivo}")

        train\_gen = AvocadoImageGenerator(train\_cases, train\_labels, batch\_size=32, augment=True)

        val\_gen = AvocadoImageGenerator(val\_cases, val\_labels, batch\_size=32, augment=False)

        train\_model(

            models[modelo\_objetivo],

            train\_gen,

            val\_gen,

            epochs=50,

            model\_name=f"{modelo\_objetivo.lower().replace(' ', '\_')}\_avocado",

            class\_weight=class\_weight

        )

%%writefile app.py

import streamlit as st

import os

import matplotlib.pyplot as plt

from model\_utils import predict\_image

from preprocessing import load\_and\_preprocess\_image

from report\_utils import generate\_pdf\_report, generate\_comparison\_report

from tensorflow.keras.models import load\_model

from metrics\_utils import evaluate\_on\_dataset

import time

import glob

import io

st.set\_page\_config(

    page\_title="Sistema de Diagnóstico de Enfermedades de Hoja de Palta",

    page\_icon="🥑",

    layout="wide"

)

st.title("Sistema Inteligente de Diagnóstico de Enfermedades de Hoja de Palta")

st.markdown("""

Este sistema utiliza redes neuronales convolucionales para analizar imágenes de hojas de palta

y detectar signos de enfermedades.

""")

MODEL\_PATHS = {

    "CNN 2D Personalizada": "models/cnn\_2d\_personalizada\_avocado.keras",

    "DenseNet121": "models/densenet121\_avocado.keras",

    "EfficientNetB0": "models/efficientnetb0\_avocado.keras"

}

@st.cache\_resource

def load\_selected\_model(model\_name):

    model\_path = MODEL\_PATHS.get(model\_name)

    if model\_path and os.path.exists(model\_path):

        return load\_model(model\_path)

    else:

        st.error(f"No se encontró el modelo: {model\_path}")

        st.stop()

st.sidebar.header("Configuración")

model\_names = list(MODEL\_PATHS.keys())

selected\_model\_name = st.sidebar.selectbox("Modelo a utilizar", model\_names)

confidence\_threshold = st.sidebar.slider(

    "Umbral de confianza para diagnóstico",

    min\_value=0.1, max\_value=0.99, value=0.5, step=0.01

)

model = load\_selected\_model(selected\_model\_name)

DATASET\_PATH = "/content/avocado\_dataset/valid"

st.header("Carga de Imagen")

upload\_option = st.radio("Seleccione el tipo de entrada", ["Subir imagen JPG/PNG", "Usar ejemplo"])

uploaded\_file = None

if upload\_option == "Subir imagen JPG/PNG":

    uploaded\_file = st.file\_uploader("Suba una imagen de hoja", type=["png", "jpg", "jpeg"])

else:

    class\_folders = [f for f in os.listdir(DATASET\_PATH) if os.path.isdir(os.path.join(DATASET\_PATH, f))]

    selected\_class = st.selectbox("Seleccione clase de ejemplo", class\_folders)

    selected\_folder = os.path.join(DATASET\_PATH, selected\_class)

    class\_images = glob.glob(os.path.join(selected\_folder, "\*.png")) + glob.glob(os.path.join(selected\_folder, "\*.jpg"))

    if class\_images:

        file\_names = [os.path.basename(f) for f in class\_images]

        file\_choice = st.selectbox("Seleccione imagen", file\_names)

        if file\_choice:

            uploaded\_file = os.path.join(selected\_folder, file\_choice)

        else:

            st.warning("Seleccione una imagen válida.")

    else:

        st.warning(f"No se encontraron imágenes en la carpeta '{selected\_class}'.")

if uploaded\_file:

    with st.spinner("Procesando imagen..."):

        image\_preprocessed, image\_original = load\_and\_preprocess\_image(uploaded\_file)

        time.sleep(1)

    st.subheader("Visualización de Imagen")

    col1, col2 = st.columns(2)

    with col1:

        st.image(image\_original, caption="Imagen Original Redimensionada", use\_container\_width=True)

    with col2:

        st.image(image\_preprocessed, caption="Imagen Preprocesada", use\_container\_width=True)

    st.header("Resultados del Diagnóstico")

    with st.spinner("Analizando imagen..."):

        prediction, confidence, heatmap = predict\_image(model, image\_preprocessed)

    col1, col2, col3 = st.columns(3)

    with col1:

        st.metric("Modelo utilizado", selected\_model\_name)

    with col2:

        st.metric("Predicción", "Enferma" if prediction == 1 else "Saludable")

    with col3:

        st.metric("Confianza", f"{confidence:.2%}")

    if confidence < confidence\_threshold:

        st.warning("La confianza es baja. Se recomienda evaluación adicional.")

    else:

        if prediction == 0:

            st.success("La hoja está saludable.")

        else:

            st.error("Se detectaron posibles signos de enfermedad en la hoja.")

    st.subheader("Mapa de Saliencia (Regiones relevantes)")

    fig, ax = plt.subplots(figsize=(6,6))

    ax.imshow(image\_original)

    ax.imshow(heatmap, cmap='jet', alpha=0.5)

    ax.axis('off')

    st.pyplot(fig)

st.subheader("Generar Reporte PDF")

if st.button("📄 Generar Reporte PDF"):

    with st.spinner("Generando reporte..."):

        pdf\_bytes = generate\_pdf\_report(

            image\_original=image\_original,

            heatmap=heatmap,

            diagnosis=prediction,

            confidence=confidence,

            model\_name=selected\_model\_name

        )

    st.download\_button(

        label="📥 Descargar Reporte PDF",

        data=pdf\_bytes,

        file\_name="reporte\_diagnostico\_avocado.pdf",

        mime="application/pdf"

    )

st.header("Comparación de Modelos con Dataset de Validación")

if st.button("Evaluar Modelos y Generar Reporte"):

    with st.spinner("Evaluando modelos..."):

        test\_dir = "/content/avocado\_dataset/valid"

        metrics\_list, comparisons, confusion\_matrices = evaluate\_on\_dataset(test\_dir)

        report\_path = generate\_comparison\_report(metrics\_list, ["CNN 2D Personalizada", "DenseNet121", "EfficientNetB0"], comparisons, confusion\_matrices)

        with open(report\_path, "rb") as f:

            st.download\_button(" 📥 Descargar Reporte de Comparación", data=f, file\_name="comparacion\_modelos.pdf", mime="application/pdf")

    st.success("Evaluación y reporte generados con éxito.")

%%writefile metrics\_utils.py

import numpy as np

from sklearn.metrics import confusion\_matrix, classification\_report

from scipy.stats import chi2

from tensorflow.keras.models import load\_model

import matplotlib.pyplot as plt

import os

from tqdm import tqdm

import seaborn as sns

from skimage.io import imread

def matthews\_corrcoef(cm):

    tp, fp, fn, tn = cm[1][1], cm[0][1], cm[1][0], cm[0][0]

    numerator = (tp \* tn) - (fp \* fn)

    denominator = np.sqrt((tp + fp) \* (tp + fn) \* (tn + fp) \* (tn + fn))

    return numerator / denominator if denominator != 0 else 0

def mcnemar\_test(y\_true, y\_model1, y\_model2):

    table = np.zeros((2, 2))

    for true, pred1, pred2 in zip(y\_true, y\_model1, y\_model2):

        if pred1 == true and pred2 != true:

            table[0][1] += 1

        elif pred1 != true and pred2 == true:

            table[1][0] += 1

    if table[0][1] + table[1][0] > 25:

        statistic = (np.abs(table[0][1] - table[1][0]) - 1) \*\* 2 / (table[0][1] + table[1][0])

    else:

        statistic = (np.abs(table[0][1] - table[1][0])) \*\* 2 / (table[0][1] + table[1][0])

    p\_value = 1 - chi2.cdf(statistic, df=1)

    return statistic, p\_value

def evaluate\_on\_dataset(test\_dir):

    def load\_data():

        cases, labels = [], []

        for label\_name, label in [('Healthy', 0), ('Disease', 1)]:

            folder = os.path.join(test\_dir, label\_name)

            for img\_file in os.listdir(folder):

                img = imread(os.path.join(folder, img\_file)).astype(np.float32) / 255.0

                cases.append(img)

                labels.append(label)

        X = np.array(cases)

        y = np.array(labels)

        return X, y

    X, y\_true = load\_data()

    models = {

        "CNN 2D Personalizada": load\_model("models/cnn\_2d\_personalizada\_avocado.keras"),

        "DenseNet121": load\_model("models/densenet121\_avocado.keras"),

        "EfficientNetB0": load\_model("models/efficientnetb0\_avocado.keras")

    }

    metrics\_list = []

    predictions\_per\_model = []

    confusion\_matrices = []

    for name, model in models.items():

        if model.input\_shape[-1] == 3 and X.shape[-1] == 1:

            X\_processed = np.repeat(X, 3, axis=-1)

        else:

            X\_processed = X

        preds = model.predict(X\_processed, batch\_size=32).flatten()

        y\_pred = (preds > 0.5).astype(int)

        predictions\_per\_model.append(y\_pred)

        cm = confusion\_matrix(y\_true, y\_pred)

        report = classification\_report(y\_true, y\_pred, target\_names=['Healthy', 'Disease'], output\_dict=True)

        metrics\_list.append({

            'model': name,

            'accuracy': report['accuracy'],

            'sensitivity': report['Disease']['recall'],

            'specificity': report['Healthy']['recall'],

            'f1': report['Disease']['f1-score'],

            'mcc': matthews\_corrcoef(cm)

        })

        plt.figure(figsize=(4, 4))

        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,

                    xticklabels=['Healthy', 'Disease'],

                    yticklabels=['Healthy', 'Disease'])

        plt.title(f'Matriz de Confusión - {name}')

        plt.xlabel('Predicho')

        plt.ylabel('Real')

        cm\_path = f'cm\_{name.replace(" ", "\_").lower()}.png'

        plt.tight\_layout()

        plt.savefig(cm\_path)

        plt.close()

        confusion\_matrices.append(cm\_path)

    comparisons = {}

    comparisons['0\_1'] = mcnemar\_test(y\_true, predictions\_per\_model[0], predictions\_per\_model[1])

    comparisons['0\_2'] = mcnemar\_test(y\_true, predictions\_per\_model[0], predictions\_per\_model[2])

    comparisons['1\_2'] = mcnemar\_test(y\_true, predictions\_per\_model[1], predictions\_per\_model[2])

    return metrics\_list, comparisons, confusion\_matrices

%%writefile report\_utils.py

from fpdf import FPDF

from PIL import Image

import numpy as np

import matplotlib.pyplot as plt

import os

import io

def generate\_pdf\_report(image\_original, heatmap, diagnosis, confidence, model\_name):

    combined\_path = "combined\_temp.png"

    fig, axes = plt.subplots(1, 2, figsize=(8, 4))

    axes[0].imshow(image\_original.astype(np.float32))

    axes[0].axis("off")

    axes[0].set\_title("Imagen Original")

    axes[1].imshow(image\_original.astype(np.float32))

    axes[1].imshow(heatmap, cmap='jet', alpha=0.5)

    axes[1].axis("off")

    axes[1].set\_title("Mapa de Saliencia")

    plt.tight\_layout()

    plt.savefig(combined\_path)

    plt.close()

    diagnosis\_text = "ENFERMA" if diagnosis == 1 else "SALUDABLE"

    pdf = FPDF()

    pdf.add\_page()

    pdf.set\_font("Arial", size=14)

    pdf.cell(200, 10, txt="Reporte de Diagnóstico de Hoja de Palta", ln=1, align='C')

    pdf.set\_font("Arial", size=12)

    pdf.ln(10)

    pdf.cell(200, 10, txt=f"Modelo utilizado: {model\_name}", ln=1)

    pdf.cell(200, 10, txt=f"Diagnóstico: {diagnosis\_text}", ln=1)

    pdf.cell(200, 10, txt=f"Confianza: {confidence:.2f}", ln=1)

    pdf.image(combined\_path, x=30, y=60, w=150)

    os.remove(combined\_path)

    # 🟢 Devuelve el contenido del PDF como bytes

    return pdf.output(dest='S').encode('latin-1')

def generate\_comparison\_report(metrics\_list, model\_names, comparisons, confusion\_matrices, output\_path="comparison\_report\_avocado.pdf"):

    """

    Genera un reporte PDF comparando métricas, pruebas estadísticas y mostrando las matrices de confusión.

    :param metrics\_list: Lista de métricas por modelo.

    :param model\_names: Lista con nombres de modelos (en orden).

    :param comparisons: Diccionario con resultados de McNemar ('0\_1', '0\_2', '1\_2').

    :param confusion\_matrices: Lista de rutas a las imágenes de las matrices de confusión.

    :param output\_path: Ruta donde guardar el PDF generado.

    """

    pdf = FPDF()

    pdf.add\_page()

    # Título

    pdf.set\_font("Arial", 'B', 16)

    pdf.cell(0, 10, txt="Reporte Comparativo de Modelos - Hoja de Palta", ln=True, align='C')

    pdf.ln(10)

    # Métricas de cada modelo

    pdf.set\_font("Arial", 'B', 12)

    pdf.cell(0, 10, txt="Métricas de Evaluación por Modelo", ln=True)

    pdf.set\_font("Arial", 'B', 10)

    headers = ['Modelo', 'Accuracy', 'Sensibilidad', 'Especificidad', 'F1-Score', 'MCC']

    col\_widths = [50, 25, 30, 30, 25, 25]

    for header, width in zip(headers, col\_widths):

        pdf.cell(width, 8, header, border=1, align='C')

    pdf.ln()

    pdf.set\_font("Arial", '', 10)

    for metrics in metrics\_list:

        pdf.cell(col\_widths[0], 8, metrics['model'], border=1)

        pdf.cell(col\_widths[1], 8, f"{metrics['accuracy']:.3f}", border=1, align='C')

        pdf.cell(col\_widths[2], 8, f"{metrics['sensitivity']:.3f}", border=1, align='C')

        pdf.cell(col\_widths[3], 8, f"{metrics['specificity']:.3f}", border=1, align='C')

        pdf.cell(col\_widths[4], 8, f"{metrics['f1']:.3f}", border=1, align='C')

        pdf.cell(col\_widths[5], 8, f"{metrics['mcc']:.3f}", border=1, align='C')

        pdf.ln()

    # Comparaciones estadísticas

    pdf.ln(10)

    pdf.set\_font("Arial", 'B', 12)

    pdf.cell(0, 10, txt="Comparaciones Estadísticas (Prueba de McNemar)", ln=True)

    pdf.set\_font("Arial", '', 10)

    # Mostrar las comparaciones usando nombres de modelos

    comparaciones\_nombres = {'0\_1': (model\_names[0], model\_names[1]),

                             '0\_2': (model\_names[0], model\_names[2]),

                             '1\_2': (model\_names[1], model\_names[2])}

    for key, (name1, name2) in comparaciones\_nombres.items():

        stat, p\_value = comparisons[key]

        pdf.cell(0, 8, txt=f"Comparación entre {name1} y {name2}:", ln=True)

        pdf.cell(0, 8, txt=f"  Estadístico McNemar: {stat:.3f} | p-valor: {p\_value:.5f}", ln=True)

    # Agregar imágenes de matrices de confusión

    pdf.add\_page()

    pdf.set\_font("Arial", 'B', 12)

    pdf.cell(0, 10, txt="Matrices de Confusión", ln=True)

    pdf.ln(5)

    for model\_name, cm\_path in zip(model\_names, confusion\_matrices):

        pdf.cell(0, 10, txt=f"Modelo: {model\_name}", ln=True)

        pdf.image(cm\_path, x=30, w=150)

        pdf.ln(10)

    pdf.output(output\_path)

    print(f"Reporte comparativo generado correctamente en: {output\_path}")

    return output\_path