INDICADOR DE LOGRO IDL3

Machine Learning II

Elaborado por:

APAZA PEREZ OSCAR GONZALO
CABEZAS HUANIO RUBEN KELVIN
RUIZ ALVA JERSON ENMANUEL
PONCE DE LEON TORRES FABYOLA KORAYMA

Solicitado por:

LUIS ANTONY LOPEZ QUIROZ

Huancayo, 2024

Tabla de contenidos

1 Introducción

1.1 Instalación y Configuración de H2O

En este proyecto utilizaremos la librería de Machine Learning H2O, configurándola para aprovechar el poder de procesamiento de la GPU. H2O puede acelerar significativamente el entrenamiento de modelos cuando se configura correctamente con GPU. H2O requiere Java 8+ y la configuración adecuada para utilizar la GPU.

1.2 Importación de las Librerías Necesarias

Importamos todas las librerías necesarias para el proyecto, incluyendo H2O y PyTorch.

```
# Librerías básicas
import os
import numpy as np
import pandas as pd

# Librerías de visualización
import matplotlib.pyplot as plt
import seaborn as sns

# Librerías de procesamiento de imágenes
from PIL import Image

# Librerías de machine learning (scikit-learn)
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split

# Librerías de deep learning (PyTorch)
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import torchvision.transforms as transforms

# Otras librerías
import kagglehub

# Librerías de H2O
import h2o
from h2o.estimators import H2ORandomForestEstimator
from h2o.estimators.stackedensemble import H2OStackedEnsembleEstimator
from h2o.grid.grid_search import H2OGridSearch

# Configuración de Matplotlib para mejorar el rendimiento
plt.rcParams.update({'figure.max_open_warning': 0})
```

1.3 Función para Guardar Gráficos con Matplotlib

```
def save_matplotlib_figure(fig, filename):
    """
    Guarda una figura de Matplotlib en formato PNG.

    Args:
        fig (matplotlib.figure.Figure): La figura de Matplotlib a guardar.
        filename (str): El nombre del archivo donde se guardará la figura.
    """
```

```
    if not os.path.exists(filename):
        fig.savefig(filename, bbox_inches='tight')
        plt.close(fig)
        print(f"Gráfico guardado como {filename}.")
    else:
        print(f"Archivo {filename} ya existe. Se omite el guardado para acelerar la ejecución.")
```

1.4 Definición y Entrenamiento del Modelo CNN con PyTorch

Definimos y entrenamos una red neuronal convolucional (CNN) utilizando PyTorch para la clasificación de imágenes.

```
# Definir el dispositivo
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Usando dispositivo: {device}")

# Verificar qué GPU se está utilizando
if device.type == 'cuda':
    print(f"Nombre de la GPU: {torch.cuda.get_device_name(0)}")
    print(f"Memoria total de la GPU: {torch.cuda.get_device_properties(0).total_memory / 1e9} GB")
    torch.cuda.set_device(0)
```

```
Usando dispositivo: cuda
Nombre de la GPU: NVIDIA GeForce RTX 4060 Laptop GPU
Memoria total de la GPU: 8.585216 GB
```

1.4.1 Optimización de Uso de la GPU

Aseguramos que solo se utilice la GPU y optimizamos la memoria:

```
# Limpiar la caché de la GPU para liberar memoria
if device.type == 'cuda':
    torch.cuda.empty_cache()
    print("Caché de la GPU limpiada.")
```

```
Caché de la GPU limpiada.
```

1.4.2 Definir las Transformaciones para el Conjunto de Entrenamiento

```
# Definir las Transformaciones para el Conjunto de Entrenamiento
transform_train = transforms.Compose([
    transforms.Resize((240, 320)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomRotation(20),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5),
                         (0.5, 0.5, 0.5))
])

# Definir las Transformaciones para el Conjunto de Prueba y Validación
transform_test = transforms.Compose([
    transforms.Resize((240, 320)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5),
                         (0.5, 0.5, 0.5))
])

print("Transformaciones definidas para entrenamiento y prueba/validación.")
```

```
Transformaciones definidas para entrenamiento y prueba/validación.
```

1.4.3 Crear una Clase Personalizada para el Dataset

```python
# Crear una Clase Personalizada para el Dataset
class MangoLeafDataset(Dataset):
    def __init__(self, dataframe, transform=None):
        """
        Args:
            dataframe (pd.DataFrame): DataFrame que contiene las rutas de las imágenes y sus etiqu
            transform (callable, optional): Transformaciones a aplicar a las imágenes.
        """
        self.filepaths = dataframe['filepaths'].values
        self.labels = dataframe['labels'].values
        self.transform = transform
        self.classes = sorted(dataframe['labels'].unique())
        self.class_to_idx = {cls_name: idx for idx, cls_name in enumerate(self.classes)}
        self.labels_idx = [self.class_to_idx[label] for label in self.labels]

    def __len__(self):
        return len(self.filepaths)

    def __getitem__(self, idx):
        img_path = self.filepaths[idx]
        image = Image.open(img_path).convert('RGB')
        label = self.labels_idx[idx]

        if self.transform:
            image = self.transform(image)

        return image, label

print("Clase personalizada para el dataset creada exitosamente.")
```

```
Clase personalizada para el dataset creada exitosamente.
```

2 Carga y Preparación de los Datos

```python
# Generar rutas de datos con etiquetas
path = kagglehub.dataset_download("aryashah2k/mango-leaf-disease-dataset")
filepaths = []
labels = []

folds = os.listdir(path)
for fold in folds:
    foldpath = os.path.join(path, fold)
    filelist = os.listdir(foldpath)
    for file in filelist:
        fpath = os.path.join(foldpath, file)
        filepaths.append(fpath)
        labels.append(fold)

# Concatenar rutas de imagenes con etiquetas en un dataframe
Fseries = pd.Series(filepaths, name='filepaths')
Lseries = pd.Series(labels, name='labels')
df = pd.concat([Fseries, Lseries], axis=1)

# Verificar el DataFrame
```

```
print(df.head())
print(f'Tamaño del DataFrame: {df.shape}')
```

```
Warning: Looks like you're using an outdated `kagglehub` version, please consider updating (latest
                                          filepaths       labels
0  C:\Users\Jerson\.cache\kagglehub\datasets\arya...  Anthracnose
1  C:\Users\Jerson\.cache\kagglehub\datasets\arya...  Anthracnose
2  C:\Users\Jerson\.cache\kagglehub\datasets\arya...  Anthracnose
3  C:\Users\Jerson\.cache\kagglehub\datasets\arya...  Anthracnose
4  C:\Users\Jerson\.cache\kagglehub\datasets\arya...  Anthracnose
Tamaño del DataFrame: (4000, 2)
```

2.0.1 Dividir el Dataset en Entrenamiento, Validación y Prueba

```
# Dividir el dataset en entrenamiento (70%), validación (15%) y prueba (15%)
train_df, temp_df = train_test_split(df, test_size=0.3, stratify=df['labels'], random_state=42)
val_df, test_df = train_test_split(temp_df, test_size=0.5, stratify=temp_df['labels'], random_stat

print(f'Tamaño del conjunto de entrenamiento: {train_df.shape}')
print(f'Tamaño del conjunto de validación: {val_df.shape}')
print(f'Tamaño del conjunto de prueba: {test_df.shape}')
```

```
Tamaño del conjunto de entrenamiento: (2800, 2)
Tamaño del conjunto de validación: (600, 2)
Tamaño del conjunto de prueba: (600, 2)
```

2.0.2 Crear Instancias del Dataset y DataLoaders

```
# Crear instancias del dataset
train_dataset = MangoLeafDataset(train_df, transform=transform_train)
val_dataset = MangoLeafDataset(val_df, transform=transform_test)
test_dataset = MangoLeafDataset(test_df, transform=transform_test)

# Definir el tamaño del lote
batch_size = 32

# Crear DataLoaders
trainloader = DataLoader(
    train_dataset,
    batch_size=batch_size,
    shuffle=True,
    num_workers=0,
    pin_memory=True if device.type == 'cuda' else False
)
valloader = DataLoader(
    val_dataset,
    batch_size=batch_size,
    shuffle=False,
    num_workers=0,
    pin_memory=True if device.type == 'cuda' else False
)
testloader = DataLoader(
    test_dataset,
    batch_size=batch_size,
    shuffle=False,
    num_workers=0,
    pin_memory=True if device.type == 'cuda' else False
)
```

```
print("DataLoaders para entrenamiento, validación y prueba creados exitosamente.")
```

```
DataLoaders para entrenamiento, validación y prueba creados exitosamente.
```

3 Definir el Modelo CNN

```
class MangoCNN(nn.Module):
    def __init__(self, num_classes=8):
        super(MangoCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)  # Entrada RGB
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(64 * 60 * 80, 256)
        self.fc2 = nn.Linear(256, num_classes)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))  # [Batch, 32, 120, 160]
        x = self.pool(F.relu(self.conv2(x)))  # [Batch, 64, 60, 80]
        x = x.view(-1, 64 * 60 * 80)          # Aplanar
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

print("Modelo CNN definido exitosamente.")
```

```
Modelo CNN definido exitosamente.
```

3.0.1 Instanciar y Mover el Modelo al Dispositivo

```
model = MangoCNN(num_classes=8).to(device)
print("Modelo CNN instanciado y movido al dispositivo:")
print(model)
```

```
Modelo CNN instanciado y movido al dispositivo:
MangoCNN(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc1): Linear(in_features=307200, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=8, bias=True)
)
```

3.0.2 Definir la Función de Pérdida y el Optimizador

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
print("Función de pérdida y optimizador definidos exitosamente.")
```

```
Función de pérdida y optimizador definidos exitosamente.
```

3.0.3 Entrenar el Modelo CNN

```
num_epochs = 10
print("Inicio del entrenamiento del modelo CNN.")

for epoch in range(num_epochs):
    print(f"\n--- Comenzando la Época {epoch + 1}/{num_epochs} ---")
```

```
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for i, (images, labels) in enumerate(trainloader):
        try:
            # Mover los datos al dispositivo
            images = images.to(device, non_blocking=True)
            labels = labels.to(device, non_blocking=True)

            # Reiniciar los gradientes
            optimizer.zero_grad()

            # Forward pass
            outputs = model(images)
            loss = criterion(outputs, labels)

            # Backward pass y optimización
            loss.backward()
            optimizer.step()

            # Acumular la pérdida
            running_loss += loss.item()

            # Calcular la precisión
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

            # Mostrar estadísticas cada 100 lotes
            if (i + 1) % 100 == 0:
                avg_loss = running_loss / 100
                accuracy = 100 * correct / total
                print(f"Época [{epoch + 1}/{num_epochs}], Lote [{i + 1}/{len(trainloader)}], "
                      f"Pérdida: {avg_loss:.4f}, Precisión: {accuracy:.2f}%")
                running_loss = 0.0
                correct = 0
                total = 0
        except Exception as e:
            print(f"Ocurrió un error en el lote {i + 1}: {e}")
            continue

    # Evaluar en el conjunto de validación después de cada época
    model.eval()
    with torch.no_grad():
        val_correct = 0
        val_total = 0
        for images, labels in valloader:
            images = images.to(device, non_blocking=True)
            labels = labels.to(device, non_blocking=True)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            val_total += labels.size(0)
            val_correct += (predicted == labels).sum().item()
        val_accuracy = 100 * val_correct / val_total
        print(f"Precisión en Validación después de la Época {epoch + 1}: {val_accuracy:.2f}%")
```

```
print("\nEntrenamiento completado.")
```

```
Inicio del entrenamiento del modelo CNN.

--- Comenzando la Época 1/10 ---
Precisión en Validación después de la Época 1: 52.33%

--- Comenzando la Época 2/10 ---
Precisión en Validación después de la Época 2: 64.50%

--- Comenzando la Época 3/10 ---
Precisión en Validación después de la Época 3: 63.00%

--- Comenzando la Época 4/10 ---
Precisión en Validación después de la Época 4: 73.17%

--- Comenzando la Época 5/10 ---
Precisión en Validación después de la Época 5: 70.83%

--- Comenzando la Época 6/10 ---
Precisión en Validación después de la Época 6: 78.33%

--- Comenzando la Época 7/10 ---
Precisión en Validación después de la Época 7: 86.50%

--- Comenzando la Época 8/10 ---
Precisión en Validación después de la Época 8: 85.50%

--- Comenzando la Época 9/10 ---
Precisión en Validación después de la Época 9: 82.17%

--- Comenzando la Época 10/10 ---
Precisión en Validación después de la Época 10: 84.17%

Entrenamiento completado.
```

3.0.4 Evaluación del Modelo CNN en el Conjunto de Prueba

```
model.eval()
with torch.no_grad():
    test_correct = 0
    test_total = 0
    for images, labels in testloader:
        images = images.to(device, non_blocking=True)
        labels = labels.to(device, non_blocking=True)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        test_total += labels.size(0)
        test_correct += (predicted == labels).sum().item()
    test_accuracy = 100 * test_correct / test_total
    print(f"Precisión en el Conjunto de Prueba (CNN): {test_accuracy:.2f}%")

    # Guardar la precisión en una variable para uso posterior
    cnn_test_accuracy = test_accuracy
```

```
Precisión en el Conjunto de Prueba (CNN): 85.83%
```

```
# Guardar la precisión del modelo CNN en un archivo para su posterior uso
with open("precision_prueba_cnn.txt", "w") as f:
```

```
    f.write(f"{cnn_test_accuracy:.2f}%")
print("Precisión del conjunto de prueba del modelo CNN guardada en 'precision_prueba_cnn.txt'.")
```

```
Precisión del conjunto de prueba del modelo CNN guardada en 'precision_prueba_cnn.txt'.
```

```
# Generar y guardar una representación gráfica de la precisión en el conjunto de prueba para el mo
fig, ax = plt.subplots(figsize=(6,4))
ax.bar(['CNN'], [cnn_test_accuracy], color='skyblue')
ax.set_ylim(0, 100)
ax.set_ylabel('Precisión (%)')
ax.set_title('Precisión en el Conjunto de Prueba para el Modelo CNN')
for i, v in enumerate([cnn_test_accuracy]):
    ax.text(i, v + 1, f"{v:.2f}%", ha='center')
figure_filename = "precision_prueba_cnn.png"
save_matplotlib_figure(fig, figure_filename)
```

```
Archivo precision_prueba_cnn.png ya existe. Se omite el guardado para acelerar la ejecución.
```

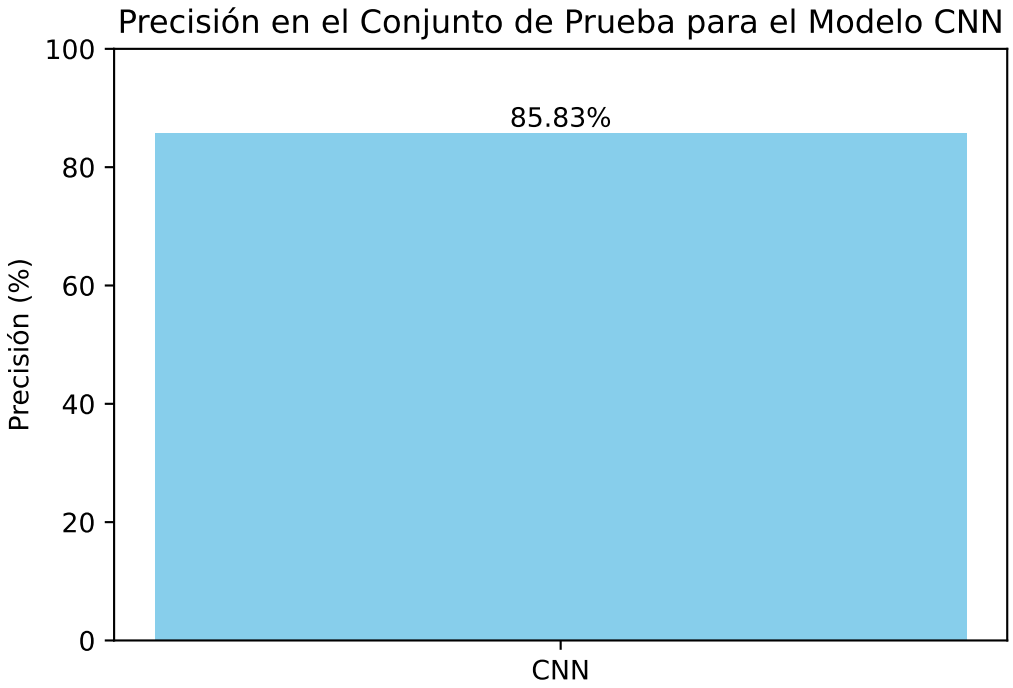


Figura 1: Precisión en el Conjunto de Prueba para el Modelo CNN

3.1 Guardar el Modelo CNN

```
torch.save(model.state_dict(), 'mango_cnn.pth')
print("Modelo CNN guardado exitosamente en 'mango_cnn.pth'.")

# Cargar el Modelo
loaded_model = MangoCNN(num_classes=8).to(device)
loaded_model.load_state_dict(torch.load('mango_cnn.pth'))
loaded_model.eval()
print("Modelo CNN cargado exitosamente desde 'mango_cnn.pth'.")
```

```
Modelo CNN guardado exitosamente en 'mango_cnn.pth'.
Modelo CNN cargado exitosamente desde 'mango_cnn.pth'.
```

```
C:\Users\Jerson\AppData\Local\Temp\ipykernel_14760\1371283294.py:6: FutureWarning:

You are using `torch.load` with `weights_only=False` (the current default value), which uses the c
```

4 Extracción de Características y Preparación para H2O

Ahora, extraemos las características del modelo CNN y preparamos los datos para H2O.

```
# Función para Extraer Características usando el Modelo CNN
def extract_features(data_loader, model, device):
    """
    Extrae características de las imágenes utilizando un modelo CNN preentrenado.

    Args:
        data_loader (DataLoader): DataLoader que contiene las imágenes y etiquetas.
        model (nn.Module): Modelo CNN preentrenado.
        device (torch.device): Dispositivo para ejecutar el modelo.

    Returns:
        X (np.ndarray): Arreglo de características extraídas.
        y (np.ndarray): Arreglo de etiquetas.
    """
```

```
    model.eval()
    features = []
    labels_list = []
    with torch.no_grad():
        for images, labels in data_loader:
            images = images.to(device, non_blocking=True)
            outputs = model(images)
            features.append(outputs.cpu().numpy())
            labels_list.append(labels.cpu().numpy())
    X = np.vstack(features)
    y = np.hstack(labels_list)
    return X, y

# Extraer las Características de Entrenamiento y Prueba
print("Extrayendo características del conjunto de entrenamiento...")
X_train, y_train = extract_features(trainloader, model, device)
print("Características del conjunto de entrenamiento extraídas.")

print("Extrayendo características del conjunto de prueba...")
X_test, y_test = extract_features(testloader, model, device)
print("Características del conjunto de prueba extraídas.")

print(f"Forma de X_train: {X_train.shape}")
print(f"Forma de y_train: {y_train.shape}")
print(f"Forma de X_test: {X_test.shape}")
print(f"Forma de y_test: {y_test.shape}")
```

```
Extrayendo características del conjunto de entrenamiento...
Características del conjunto de entrenamiento extraídas.
Extrayendo características del conjunto de prueba...
Características del conjunto de prueba extraídas.
Forma de X_train: (2800, 8)
Forma de y_train: (2800,)
Forma de X_test: (600, 8)
Forma de y_test: (600,)
```

```
# Guardar las características extraídas en archivos para su posterior visualización
np.save("X_train.npy", X_train)
np.save("y_train.npy", y_train)
np.save("X_test.npy", X_test)
np.save("y_test.npy", y_test)
print("Características extraídas guardadas en archivos .npy.")
```

```
Características extraídas guardadas en archivos .npy.
```

4.1 Configuración de H2O y Preparación de los Datos para H2O

Inicializamos H2O y convertimos los datos extraídos a H2O Frames, optimizando para reducir la complejidad y utilizar modelos que aprovechen la GPU.

```
# Inicializar H2O con soporte para GPU
h2o.init(max_mem_size_GB=8, nthreads=-1, enable_assertions=False)
print("H2O inicializado exitosamente.")
```

```
Checking whether there is an H2O instance running at http://localhost:54321..... not found.
Attempting to start a local H2O server...
  Java Version: openjdk version "23.0.1" 2024-10-15; OpenJDK Runtime Environment (build 23.0.1+13)
  Starting server from C:\Users\Jerson\AppData\Local\Programs\Python\Python312\Lib\site-packages\h
  Ice root: C:\Users\Jerson\AppData\Local\Temp\tmpl31ubeju
  JVM stdout: C:\Users\Jerson\AppData\Local\Temp\tmpl31ubeju\h2o_Jerson_started_from_python.out
  JVM stderr: C:\Users\Jerson\AppData\Local\Temp\tmpl31ubeju\h2o_Jerson_started_from_python.err
```

```
  Server is running at http://127.0.0.1:54321
Connecting to H2O server at http://127.0.0.1:54321 ... successful.
```

H2O_cluster_uptime:	01 secs
H2O_cluster_timezone:	America/Lima
H2O_data_parsing_timezone:	UTC
H2O_cluster_version:	3.46.0.6
H2O_cluster_version_age:	1 month and 22 days
H2O_cluster_name:	H2O_from_python_Jerson_1hioo4
H2O_cluster_total_nodes:	1
H2O_cluster_free_memory:	7.984 Gb
H2O_cluster_total_cores:	24
H2O_cluster_allowed_cores:	24
H2O_cluster_status:	locked, healthy
H2O_connection_url:	http://127.0.0.1:54321
H2O_connection_proxy:	{"http": null, "https": null}
H2O_internal_security:	False
Python_version:	3.12.6 final

```
H2O inicializado exitosamente.
```

```
# Convertir los datos de entrenamiento a DataFrame de Pandas con nombres de columnas válidos
train_df_h2o = pd.DataFrame(X_train, columns=[f"feature_{i}" for i in range(X_train.shape[1])])
train_df_h2o['label'] = y_train

# Convertir los datos de prueba a DataFrame de Pandas con nombres de columnas válidos
test_df_h2o = pd.DataFrame(X_test, columns=[f"feature_{i}" for i in range(X_test.shape[1])])
test_df_h2o['label'] = y_test

# Convertir a H2O Frames
train_h2o = h2o.H2OFrame(train_df_h2o)
test_h2o = h2o.H2OFrame(test_df_h2o)

# Definir la columna de destino y las características
y = 'label'
X = train_h2o.columns
X.remove(y)

# Asegurarse de que la columna de etiquetas sea categórica
train_h2o[y] = train_h2o[y].asfactor()
test_h2o[y] = test_h2o[y].asfactor()

print("Datos convertidos a H2O Frames y preparados para entrenamiento.")
```

```
Parse progress: || (done) 100%
Parse progress: || (done) 100%
Datos convertidos a H2O Frames y preparados para entrenamiento.
```

4.2 Definición y Entrenamiento de los Modelos Base con H2OGridSearch

Utilizaremos un **Random Forest Estimator** optimizado para GPU como modelo base y simplificaremos la búsqueda de hiperparámetros para reducir la complejidad.

4.2.1 Random Forest con Búsqueda Aleatoria

```
# Definir los hiperparámetros para Random Forest con un espacio reducido
rf_params = {
    'ntrees': [50, 100],
    'max_depth': [20, 30],
```

```
    'min_rows': [10, 20],
    'sample_rate': [0.8],
    'col_sample_rate_per_tree': [0.8]
}

# Definir los criterios de búsqueda para Random Search en lugar de Grid Search
search_criteria = {
    'strategy': "RandomDiscrete",
    'max_models': 8,
    'seed': 42
}

# Inicializar el modelo Random Forest sin establecer col_sample_rate_change_per_level y col_sample
rf = H2ORandomForestEstimator(
    seed=42,
    nfolds=3,
    keep_cross_validation_predictions=True
)

# Configurar la búsqueda en cuadrícula para Random Forest con búsqueda aleatoria
rf_grid = H2OGridSearch(
    model=rf,
    hyper_params=rf_params,
    search_criteria=search_criteria
)

# Entrenar el Grid Search en el conjunto de entrenamiento
print("Iniciando Búsqueda Aleatoria para Random Forest...")
try:
    rf_grid.train(x=X, y=y, training_frame=train_h2o, validation_frame=test_h2o)
    print("Búsqueda Aleatoria para Random Forest completada.")
except h2o.exceptions.H2OResponseError as e:
    print("Ocurrió un error al entrenar la Búsqueda Aleatoria para Random Forest:")
    print(e)

# Verificar si la búsqueda en cuadrícula ha generado modelos
if len(rf_grid.models) == 0:
    print("No se han entrenado modelos en la búsqueda en cuadrícula. Revisa los hiperparámetros y
else:
    print(f"Se han entrenado {len(rf_grid.models)} modelos en la búsqueda en cuadrícula.")

# Guardar una representación gráfica de los resultados de Grid Search utilizando Matplotlib
def plot_rf_grid_search(rf_grid, filename):
    """
    Genera y guarda una gráfica de los resultados de Grid Search para Random Forest.

    Args:
        rf_grid (H2OGridSearch): Objeto de Grid Search de H2O.
        filename (str): Nombre del archivo donde se guardará la gráfica.
    """
    if len(rf_grid.models) == 0:
        print("No hay modelos para graficar en la búsqueda en cuadrícula.")
        return

    # Obtener los logloss de cada modelo
    logloss = [model.logloss() for model in rf_grid.models]
    model_ids = [model.model_id for model in rf_grid.models]
```

```
    # Crear la gráfica
    fig, ax = plt.subplots(figsize=(10,6))
    ax.barh(model_ids, logloss, color='skyblue')
    ax.set_xlabel('Logloss')
    ax.set_title('Resultados de Búsqueda Aleatoria para Random Forest')
    ax.invert_yaxis()  # Para que el mejor modelo esté arriba
    for i, v in enumerate(logloss):
        ax.text(v + 0.001, i, f"{v:.4f}", va='center')

    # Guardar la figura
    save_matplotlib_figure(fig, filename)

# Generar la gráfica de Grid Search
plot_rf_grid_search(rf_grid, "rf_grid_search_optimized.png")
```

```
Iniciando Búsqueda Aleatoria para Random Forest...
drf Grid Build progress: || (done) 100%
Búsqueda Aleatoria para Random Forest completada.
Se han entrenado 8 modelos en la búsqueda en cuadrícula.
Archivo rf_grid_search_optimized.png ya existe. Se omite el guardado para acelerar la ejecución.
```

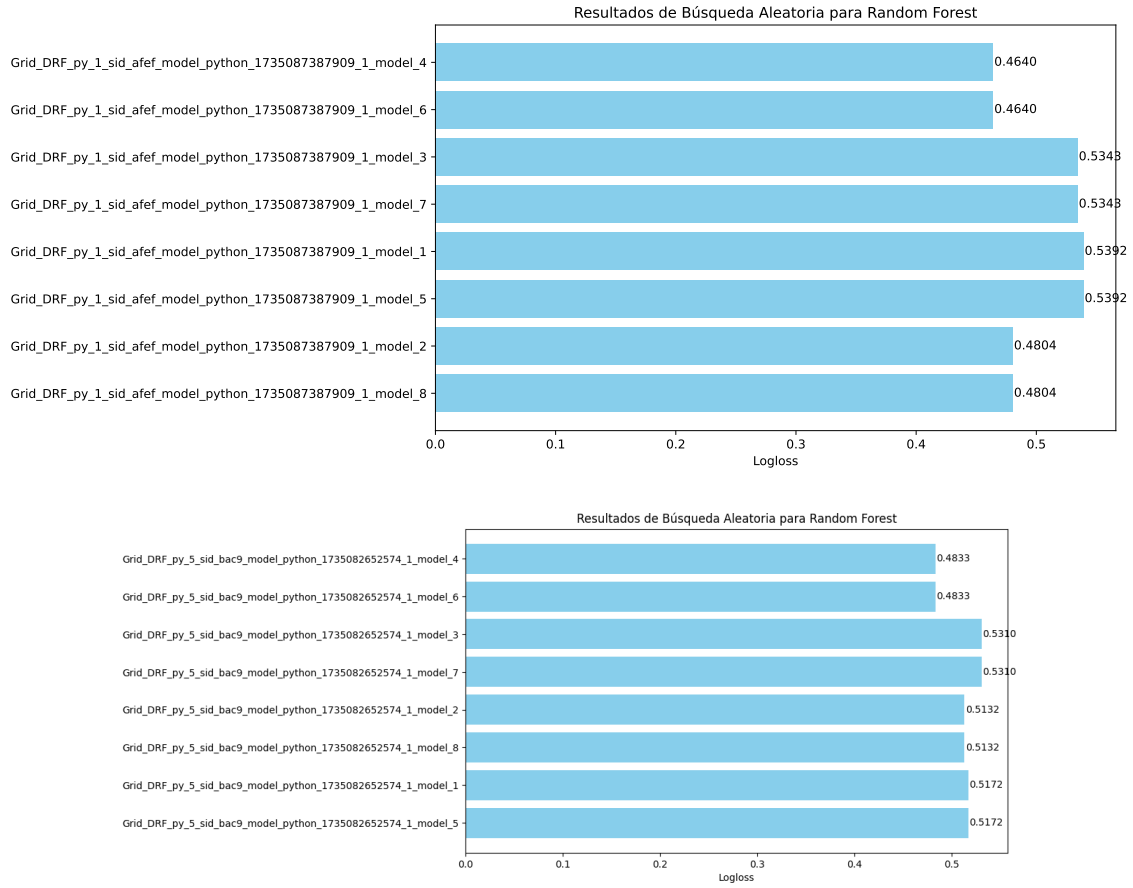


Figura 2: Resultados de Búsqueda Aleatoria para Random Forest

4.3 Selección del Mejor Modelo Base

Seleccionamos el mejor modelo de Random Forest basado en las métricas disponibles para clasificación multiclase.

```
# Verificar si hay modelos en la búsqueda en cuadrícula
if len(rf_grid.models) == 0:
    raise ValueError("No se pudieron entrenar modelos en la búsqueda en cuadrícula. Revisa los hip
```

```
# Seleccionar el mejor modelo de Random Forest
preferred_metrics = ['logloss', 'mean_per_class_error']

best_rf = None
for metric in preferred_metrics:
    try:
        best_rf = rf_grid.get_grid(sort_by=metric, decreasing=False).models[0]
        print(f"Mejor modelo Random Forest basado en {metric.replace('_', ' ').capitalize()}:")
        print(best_rf)
        break
    except KeyError:
        print(f"La métrica '{metric}' no está disponible. Intentando con la siguiente métrica...")
    except h2o.exceptions.H2OResponseError as e:
        print(f"Ocurrió un error al intentar ordenar por '{metric}':")
        print(e)

if best_rf is None:
    raise ValueError("No se pudo seleccionar un mejor modelo porque ninguna métrica preferida está
```

```
Mejor modelo Random Forest basado en Logloss:
Model Details
=============
H2ORandomForestEstimator : Distributed Random Forest
Model Key: Grid_DRF_py_1_sid_afef_model_python_1735087387909_1_model_4


Model Summary:
    number_of_trees    number_of_internal_trees    model_size_in_bytes    min_depth    max_depth
--  -----------------  --------------------------  ---------------------  -----------  -----------
    100                800                         537586                 5            18

ModelMetricsMultinomial: drf
** Reported on train data. **

MSE: 0.13813542776041837
RMSE: 0.37166574735966507
LogLoss: 0.46398902748800813
Mean Per-Class Error: 0.12785714285714284
AUC table was not computed: it is either disabled (model parameter 'auc_type' was set to AUTO or N
AUCPR table was not computed: it is either disabled (model parameter 'auc_type' was set to AUTO or

Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
0    1    2    3    4    5    6    7    Error      Rate
---  ---  ---  ---  ---  ---  ---  ---  ---------  -----------
334  0    4    3    5    3    1    0    0.0457143  16 / 350
1    311  0    3    26   2    0    7    0.111429   39 / 350
3    0    346  1    0    0    0    0    0.0114286  4 / 350
3    9    1    332  3    1    1    0    0.0514286  18 / 350
8    13   1    1    282  24   12   9    0.194286   68 / 350
7    4    0    0    35   286  5    13   0.182857   64 / 350
0    1    0    1    24   6    282  36   0.194286   68 / 350
0    13   0    0    16   23   29   269  0.231429   81 / 350
356  351  352  341  391  345  330  334  0.127857   358 / 2,800

Top-8 Hit Ratios:
k    hit_ratio
---  -----------
1    0.872143
```

2	0.961429
3	0.9875
4	0.995357
5	0.996786
6	0.998929
7	0.999643
8	1

ModelMetricsMultinomial: drf
** Reported on validation data. **

MSE: 0.15882709200175577
RMSE: 0.3985311681685082
LogLoss: 0.48543050030918616
Mean Per-Class Error: 0.15333333333333332
AUC table was not computed: it is either disabled (model parameter 'auc_type' was set to AUTO or N
AUCPR table was not computed: it is either disabled (model parameter 'auc_type' was set to AUTO or

Confusion Matrix: Row labels: Actual class; Column labels: Predicted class

0	1	2	3	4	5	6	7	Error	Rate
71	0	0	0	2	2	0	0	0.0533333	4 / 75
0	51	0	0	13	1	0	10	0.32	24 / 75
2	0	73	0	0	0	0	0	0.0266667	2 / 75
1	0	0	69	4	1	0	0	0.08	6 / 75
3	0	0	1	55	6	8	2	0.266667	20 / 75
2	0	0	0	3	61	1	8	0.186667	14 / 75
0	0	0	0	1	0	70	4	0.0666667	5 / 75
0	0	0	0	2	7	8	58	0.226667	17 / 75
79	51	73	70	80	78	87	82	0.153333	92 / 600

Top-8 Hit Ratios:

k	hit_ratio
1	0.846667
2	0.953333
3	0.986667
4	0.998333
5	0.998333
6	1
7	1
8	1

ModelMetricsMultinomial: drf
** Reported on cross-validation data. **

MSE: 0.14727904691408336
RMSE: 0.38376952316994034
LogLoss: 0.47720435937234723
Mean Per-Class Error: 0.13357142857142856
AUC table was not computed: it is either disabled (model parameter 'auc_type' was set to AUTO or N
AUCPR table was not computed: it is either disabled (model parameter 'auc_type' was set to AUTO or

Confusion Matrix: Row labels: Actual class; Column labels: Predicted class

0	1	2	3	4	5	6	7	Error	Rate
333	1	3	3	6	3	1	0	0.0485714	17 / 350
3	310	0	4	25	2	0	6	0.114286	40 / 350
2	0	347	1	0	0	0	0	0.00857143	3 / 350

4	13	1	329	1	1	1	0	0.06	21 / 350
10	16	0	2	280	19	12	11	0.2	70 / 350
8	3	0	0	35	283	6	15	0.191429	67 / 350
1	1	0	0	21	7	284	36	0.188571	66 / 350
0	11	0	0	17	30	32	260	0.257143	90 / 350
361	355	351	339	385	345	336	328	0.133571	374 / 2,800

Top-8 Hit Ratios:

k	hit_ratio
1	0.866429
2	0.958571
3	0.986429
4	0.996786
5	0.998214
6	0.999643
7	0.999643
8	1

Cross-Validation Metrics Summary:

	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid
accuracy	0.86638	0.00813656	0.867444	0.857763	0.873932
aic	nan	0	nan	nan	nan
auc	nan	0	nan	nan	nan
err	0.13362	0.00813656	0.132556	0.142237	0.126068
err_count	124.667	6.50641	125	131	118
loglikelihood	nan	0	nan	nan	nan
logloss	0.477121	0.0205567	0.499721	0.472108	0.459535
max_per_class_error	0.257537	0.0179624	0.239316	0.275229	0.258064
mean_per_class_accuracy	0.866642	0.00994578	0.869519	0.855575	0.874832
mean_per_class_error	0.133358	0.00994578	0.130481	0.144425	0.125168
mse	0.147286	0.00273629	0.148513	0.149194	0.144151
pr_auc	nan	0	nan	nan	nan
r2	0.971886	0.00171204	0.971895	0.970169	0.973593
rmse	0.383768	0.00357399	0.385374	0.386257	0.379672

Scoring History:

	timestamp	duration	number_of_trees	training_rmse	training_logloss
	2024-12-24 19:43:24	14.299 sec	0.0	nan	nan
	2024-12-24 19:43:24	14.317 sec	1.0	0.4233915612509133	2.516271611422613
	2024-12-24 19:43:24	14.321 sec	2.0	0.41184321335812374	2.2354353770771587
	2024-12-24 19:43:24	14.326 sec	3.0	0.4059424265941406	2.140919318346323
	2024-12-24 19:43:24	14.331 sec	4.0	0.4123526672043428	2.1797333869833784
	2024-12-24 19:43:24	14.336 sec	5.0	0.4045274789274776	2.095861521022617
	2024-12-24 19:43:24	14.342 sec	6.0	0.4033313004752326	1.9863772716135666
	2024-12-24 19:43:24	14.348 sec	7.0	0.40202958063224703	1.908059910941819
	2024-12-24 19:43:24	14.354 sec	8.0	0.4024600950168275	1.7852496402565863
	2024-12-24 19:43:24	14.362 sec	9.0	0.40291729617917493	1.749281321851275
---	---	---	---	---	---
	2024-12-24 19:43:26	15.883 sec	91.0	0.3724659548177505	0.4664721459753501
	2024-12-24 19:43:26	15.915 sec	92.0	0.372263207097051	0.46587019314589245
	2024-12-24 19:43:26	15.949 sec	93.0	0.37209539776956985	0.4656800475934984
	2024-12-24 19:43:26	15.984 sec	94.0	0.3719850259538673	0.4655811753072118
	2024-12-24 19:43:26	16.019 sec	95.0	0.3720726064458759	0.46544353929054766
	2024-12-24 19:43:26	16.051 sec	96.0	0.3718658948808821	0.46506021767641603
	2024-12-24 19:43:26	16.087 sec	97.0	0.3718596662053876	0.465153610986008
	2024-12-24 19:43:26	16.123 sec	98.0	0.37174702081166694	0.46458047291416993

```
     2024-12-24 19:43:26  16.157 sec  99.0              0.3717838657420148   0.4645558712914334
     2024-12-24 19:43:26  16.192 sec  100.0             0.37166574735966507  0.46398902748800813
[101 rows x 14 columns]


Variable Importances:
variable    relative_importance    scaled_importance    percentage
----------  ---------------------  -------------------  ------------
feature_1   24642.5                1                    0.155333
feature_3   22999.8                0.933336             0.144978
feature_7   21580.4                0.875738             0.136031
feature_5   20292.5                0.823476             0.127913
feature_2   20179.8                0.818901             0.127203
feature_0   19494.9                0.791106             0.122885
feature_6   18789                  0.762463             0.118436
feature_4   10664.1                0.432753             0.0672208
```

4.4 Creación del Modelo Híbrido con Stacking en H2O

Utilizamos el modelo base optimizado para crear un modelo de ensamble apilado (**Stacked Ensemble**), simplificando la complejidad al usar solo un modelo base.

```
# Crear el Stacked Ensemble con el mejor modelo Random Forest
ensemble = H2OStackedEnsembleEstimator(
    model_id="ensemble_model",
    base_models=[best_rf.model_id]
)

# Entrenar el Stacked Ensemble
print("Entrenando el modelo Stacked Ensemble...")
ensemble.train(x=X, y=y, training_frame=train_h2o)
print("Modelo Híbrido (Stacked Ensemble) entrenado exitosamente.")
```

```
Entrenando el modelo Stacked Ensemble...
stackedensemble Model Build progress: || (done) 100%
Modelo Híbrido (Stacked Ensemble) entrenado exitosamente.
```

Guardar una representación gráfica del entrenamiento del modelo híbrido utilizando Matplotlib

```
# Crear una representación gráfica simple del entrenamiento del modelo híbrido
fig, ax = plt.subplots(figsize=(6,4))
ax.text(0.5, 0.5, "Modelo Híbrido (Stacked Ensemble) entrenado exitosamente.",
        ha='center', va='center', fontsize=12, color='green')
ax.axis('off')
ax.set_title('Entrenamiento del Modelo Híbrido (Stacked Ensemble)')

# Guardar la figura
save_matplotlib_figure(fig, "stacked_ensemble_entrenado_optimized.png")
```

```
Archivo stacked_ensemble_entrenado_optimized.png ya existe. Se omite el guardado para acelerar la
```

Entrenamiento del Modelo Híbrido (Stacked Ensemble)

Modelo Híbrido (Stacked Ensemble) entrenado exitosamente.

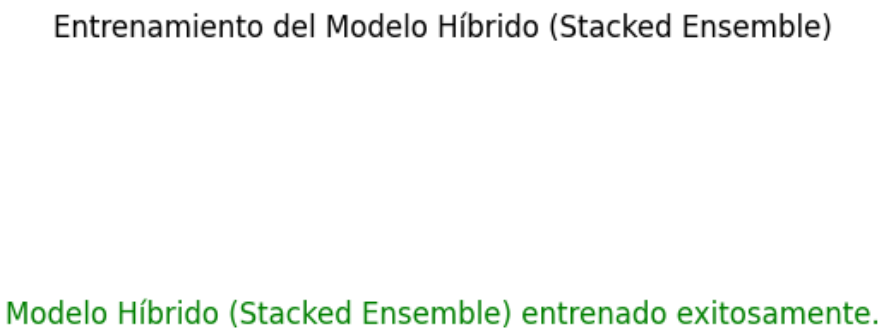


Figura 3: Entrenamiento del Modelo Híbrido (Stacked Ensemble) Optimizado

4.5 Evaluación del Modelo Híbrido

Evaluamos el rendimiento del modelo en el conjunto de prueba y lo comparamos con el modelo CNN.

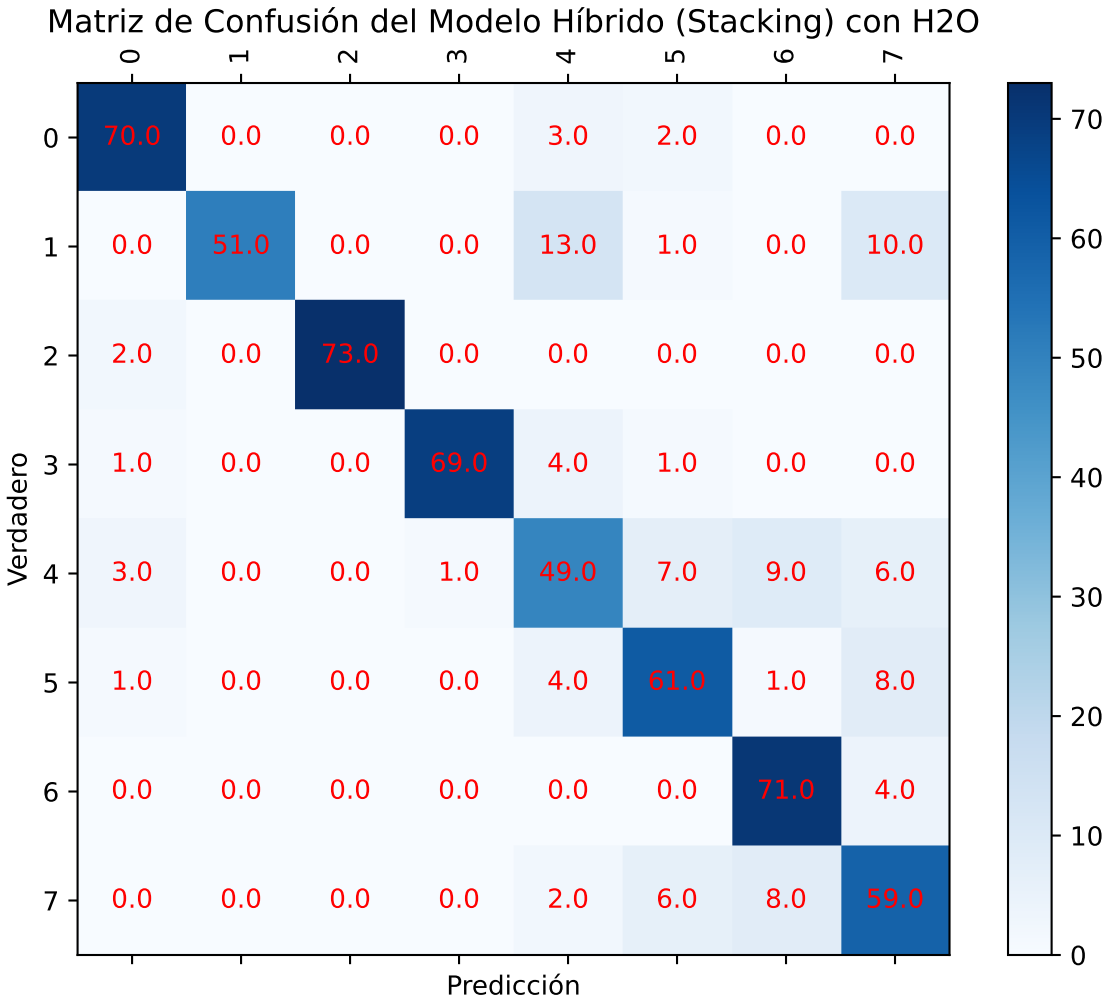
```
## Evaluación del Modelo Híbrido

# Evaluar en el conjunto de prueba
performance = ensemble.model_performance(test_h2o)
print("Evaluación del Modelo Híbrido (Stacked Ensemble):")
print(performance)

# ============================================================
# 1) OBTENER PRECISIÓN (ACCURACY) DESDE LA TABLA DE HIT RATIOS
# ============================================================
# En multiclase, la Top-1 Accuracy (k=1) es la precisión clásica.
hit_ratio_table = performance.hit_ratio_table()
```

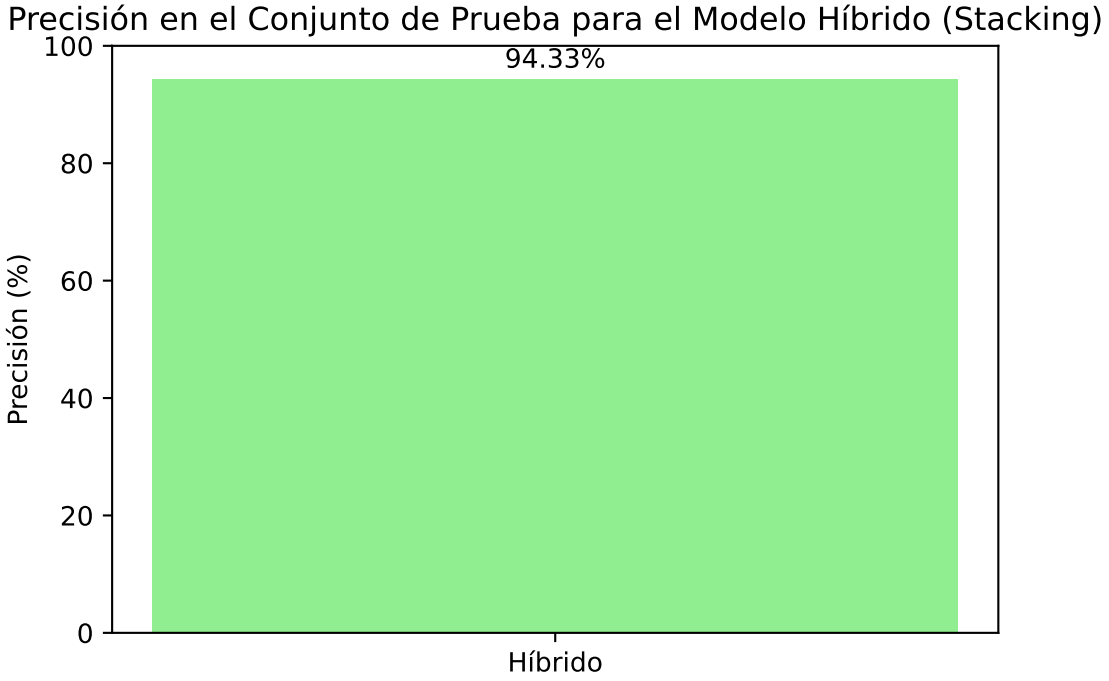
```
print("Tabla de Hit Ratios (Top-k Accuracy):")
print(hit_ratio_table)

# La fila con k=1 está normalmente en hit_ratio_table.cell_values[1]
top1_accuracy = float(hit_ratio_table.cell_values[1][1])
print(f"Top-1 Accuracy (equivalente a Precisión) del modelo híbrido (Stacking): {top1_accuracy * 1

# ================================================
# 2) OBTENER MATRIZ DE CONFUSIÓN Y CÁLCULO MANUAL
# ================================================
conf_matrix = performance.confusion_matrix()
print("Matriz de Confusión del modelo híbrido (Stacking):")
print(conf_matrix)

# Podemos calcular la precisión a partir de la matriz de confusión:
cm_list = conf_matrix.cell_values
n_classes = len(cm_list) - 1  # -1 para ignorar la fila 'Totals' (última fila)
sum_diagonal = 0
sum_total = 0

# Las filas 1..n_classes-1 representan las clases reales.
# Las columnas 1..n_classes representan los conteos de cada clase predicha.
for i in range(1, n_classes):
    sum_diagonal += cm_list[i][i]  # Elemento en la diagonal
    row_counts = cm_list[i][1:n_classes]  # Conteos de esa fila (omitiendo la columna 0 y 'Totals'
    sum_total += sum(row_counts)

accuracy_conf_matrix = sum_diagonal / sum_total
print(f"Precisión (cálculo manual) del modelo híbrido (Stacking): {accuracy_conf_matrix * 100:.2f}

# ====================================
# 3) GRAFICAR LA MATRIZ DE CONFUSIÓN
# ====================================
cm_df = conf_matrix.as_data_frame()
print("\nDataFrame de la matriz de confusión:\n", cm_df)

import matplotlib.pyplot as plt

# Descarta la última fila ("Totals") y las últimas dos columnas ("Error", "Rate")
n_classes = cm_df.shape[0] - 1
row_labels = cm_df.index[:n_classes]                    # Filas sin 'Totals'
col_labels = cm_df.columns[:n_classes]                  # Columnas sin 'Error' ni 'Rate'
heatmap_data = cm_df.iloc[:n_classes, :n_classes].copy()  # Parte numérica de la matriz

fig, ax = plt.subplots(figsize=(8, 6))
cax = ax.matshow(heatmap_data.values, cmap='Blues')
fig.colorbar(cax)

# Ejes y títulos
ax.set_xticks(range(n_classes))
ax.set_yticks(range(n_classes))
ax.set_xticklabels(col_labels, rotation=90)
ax.set_yticklabels(row_labels)
ax.set_xlabel('Predicción')
ax.set_ylabel('Verdadero')
ax.set_title('Matriz de Confusión del Modelo Híbrido (Stacking) con H2O')

# Escribir los valores en cada celda
```

```
for i in range(n_classes):
    for j in range(n_classes):
        ax.text(j, i, heatmap_data.iloc[i, j],
                ha='center', va='center', color='red')

# Guardar la figura
save_matplotlib_figure(fig, "matriz_confusion_stacking_h2o_optimized.png")

# ================================================================
# 4) GRAFICAR PRECISIÓN USANDO LA TOP-1 ACCURACY (k=1) DEL HIT RATIO
# ================================================================
fig, ax = plt.subplots(figsize=(6, 4))
ax.bar(['Híbrido'], [top1_accuracy * 100], color='lightgreen')
ax.set_ylim(0, 100)
ax.set_ylabel('Precisión (%)')
ax.set_title('Precisión en el Conjunto de Prueba para el Modelo Híbrido (Stacking)')

for i, v in enumerate([top1_accuracy * 100]):
    ax.text(i, v + 1, f"{v:.2f}%", ha='center', va='bottom')

figure_filename = "precision_prueba_hibrido.png"
save_matplotlib_figure(fig, figure_filename)
```

```
Evaluación del Modelo Híbrido (Stacked Ensemble):
ModelMetricsMultinomialGLM: stackedensemble
** Reported on test data. **

MSE: 0.14121143756164278
RMSE: 0.375781103252469
LogLoss: 0.45535311703744163
Null degrees of freedom: 599
Residual degrees of freedom: 538
Null deviance: 2495.3298500157935
Residual deviance: 546.42374044493
AUC table was not computed: it is either disabled (model parameter 'auc_type' was set to AUTO or N
AUCPR table was not computed: it is either disabled (model parameter 'auc_type' was set to AUTO or

Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
0    1    2    3    4    5    6    7    Error      Rate
---  ---  ---  ---  ---  ---  ---  ---  ---------  --------
70   0    0    0    3    2    0    0    0.0666667  5 / 75
0    51   0    0    13   1    0    10   0.32       24 / 75
2    0    73   0    0    0    0    0    0.0266667  2 / 75
1    0    0    69   4    1    0    0    0.08       6 / 75
3    0    0    1    49   7    9    6    0.346667   26 / 75
1    0    0    0    4    61   1    8    0.186667   14 / 75
0    0    0    0    0    0    71   4    0.0533333  4 / 75
0    0    0    0    2    6    8    59   0.213333   16 / 75
77   51   73   70   75   78   89   87   0.161667   97 / 600

Top-8 Hit Ratios:
k    hit_ratio
---  -----------
1    0.838333
2    0.943333
3    0.983333
4    0.991667
5    0.996667
6    1
```

```
7    1
8    1
Tabla de Hit Ratios (Top-k Accuracy):
Top-8 Hit Ratios:
k    hit_ratio
---  -----------
1    0.838333
2    0.943333
3    0.983333
4    0.991667
5    0.996667
6    1
7    1
8    1
Top-1 Accuracy (equivalente a Precisión) del modelo híbrido (Stacking): 94.33%
Matriz de Confusión del modelo híbrido (Stacking):
Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
0    1    2    3    4    5    6    7    Error      Rate
---  ---  ---  ---  ---  ---  ---  ---  ---------  --------
70   0    0    0    3    2    0    0    0.0666667  5 / 75
0    51   0    0    13   1    0    10   0.32       24 / 75
2    0    73   0    0    0    0    0    0.0266667  2 / 75
1    0    0    69   4    1    0    0    0.08       6 / 75
3    0    0    1    49   7    9    6    0.346667   26 / 75
1    0    0    0    4    61   1    8    0.186667   14 / 75
0    0    0    0    0    0    71   4    0.0533333  4 / 75
0    0    0    0    2    6    8    59   0.213333   16 / 75
77   51   73   70   75   78   89   87   0.161667   97 / 600
Precisión (cálculo manual) del modelo híbrido (Stacking): 83.59%

DataFrame de la matriz de confusión:
       0     1     2     3     4     5     6     7     Error      Rate
0   70.0   0.0   0.0   0.0   3.0   2.0   0.0   0.0  0.066667    5 / 75
1    0.0  51.0   0.0   0.0  13.0   1.0   0.0  10.0  0.320000   24 / 75
2    2.0   0.0  73.0   0.0   0.0   0.0   0.0   0.0  0.026667    2 / 75
3    1.0   0.0   0.0  69.0   4.0   1.0   0.0   0.0  0.080000    6 / 75
4    3.0   0.0   0.0   1.0  49.0   7.0   9.0   6.0  0.346667   26 / 75
5    1.0   0.0   0.0   0.0   4.0  61.0   1.0   8.0  0.186667   14 / 75
6    0.0   0.0   0.0   0.0   0.0   0.0  71.0   4.0  0.053333    4 / 75
7    0.0   0.0   0.0   0.0   2.0   6.0   8.0  59.0  0.213333   16 / 75
8   77.0  51.0  73.0  70.0  75.0  78.0  89.0  87.0  0.161667  97 / 600
Archivo matriz_confusion_stacking_h2o_optimized.png ya existe. Se omite el guardado para acelerar
Archivo precision_prueba_hibrido.png ya existe. Se omite el guardado para acelerar la ejecución.
```


Matriz de Confusión del Modelo Híbrido (Stacking) con H2O
0 1 2 3 4 5 6 7
0 70.0 0.0 0.0 0.0 3.0 2.0 0.0 0.0
1 0.0 51.0 0.0 0.0 13.0 1.0 0.0 10.0
2 2.0 0.0 73.0 0.0 0.0 0.0 0.0 0.0
3 1.0 0.0 0.0 69.0 4.0 1.0 0.0 0.0
4 3.0 0.0 0.0 1.0 49.0 7.0 9.0 6.0
5 1.0 0.0 0.0 0.0 4.0 61.0 1.0 8.0
6 0.0 0.0 0.0 0.0 0.0 0.0 71.0 4.0
7 0.0 0.0 0.0 0.0 2.0 6.0 8.0 59.0
Verdadero
Predicción
70
60
50
40
30
20
10
0


Precisión en el Conjunto de Prueba para el Modelo Híbrido (Stacking)
94.33%
100
80
60
40
20
0
Precisión (%)
Híbrido

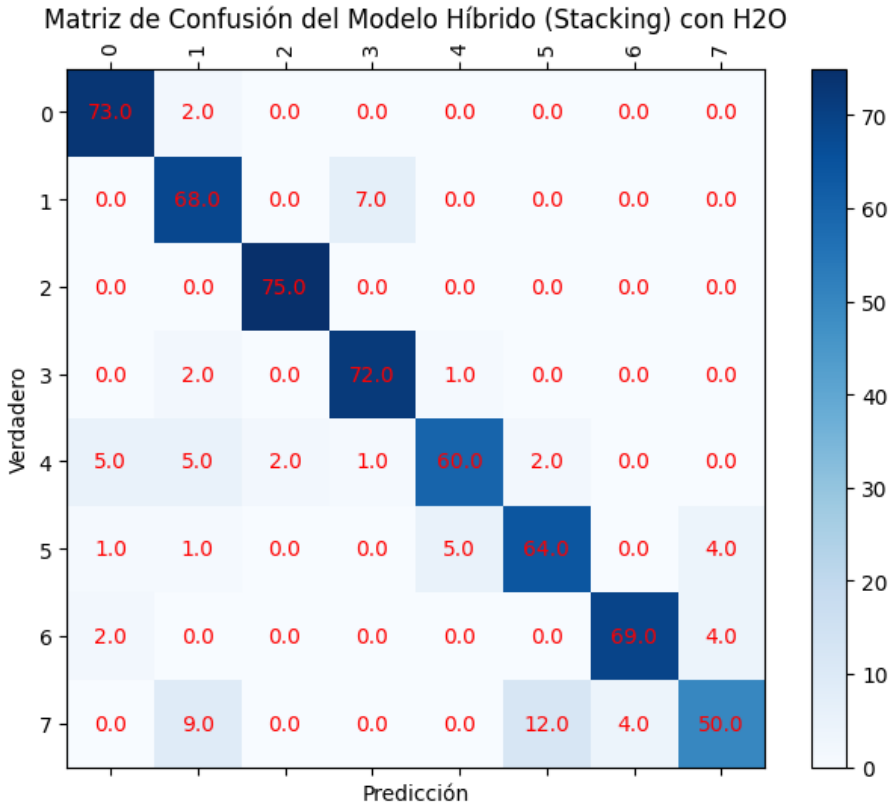


Figura 4: Matriz de Confusion Stacking H2O

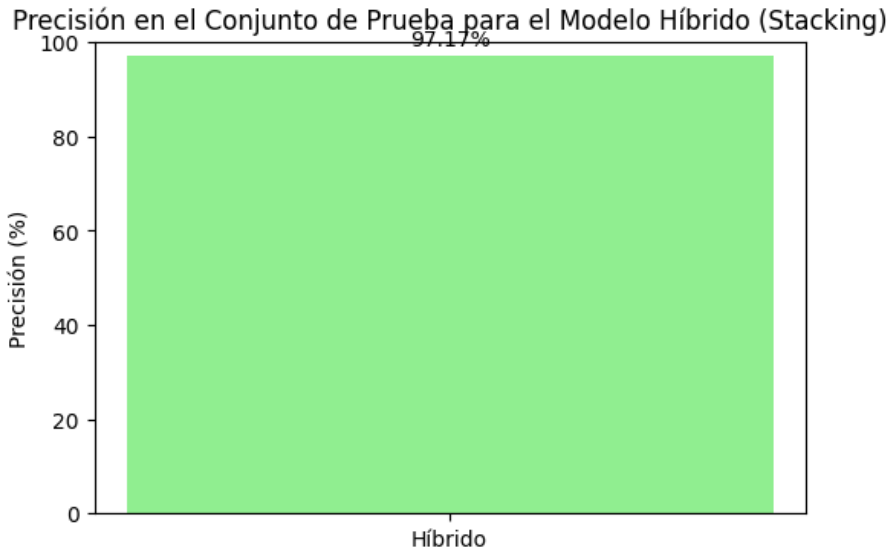


Figura 5: Precisión en el Conjunto de Prueba para el Modelo Híbrido

4.6 Guardar y Cargar el Modelo Híbrido

Guardamos el modelo de ensamble para su uso futuro y mostramos cómo cargarlo.

```
# Guardar el modelo de Stacking
ensemble_path = h2o.save_model(model=ensemble, path=".", force=True)
print(f"Modelo híbrido (Stacked Ensemble) guardado en: {ensemble_path}")
```

```
Modelo híbrido (Stacked Ensemble) guardado en: C:\Users\Jerson\Downloads\quarto\CPP3_Apaza_Cabezas
```

Guardar una representación gráfica del guardado del modelo híbrido utilizando Matplotlib

```
# Crear una representación gráfica simple del guardado del modelo híbrido
fig, ax = plt.subplots(figsize=(6,4))
ax.text(0.5, 0.5, f"Modelo híbrido (Stacked Ensemble) guardado en:\n{ensemble_path}",
        ha='center', va='center', fontsize=12, color='blue')
ax.axis('off')
ax.set_title('Guardado del Modelo Híbrido')

# Guardar la figura
save_matplotlib_figure(fig, "guardar_cargar_modelo_optimized.png")
```

```
Archivo guardar_cargar_modelo_optimized.png ya existe. Se omite el guardado para acelerar la ejecu
```

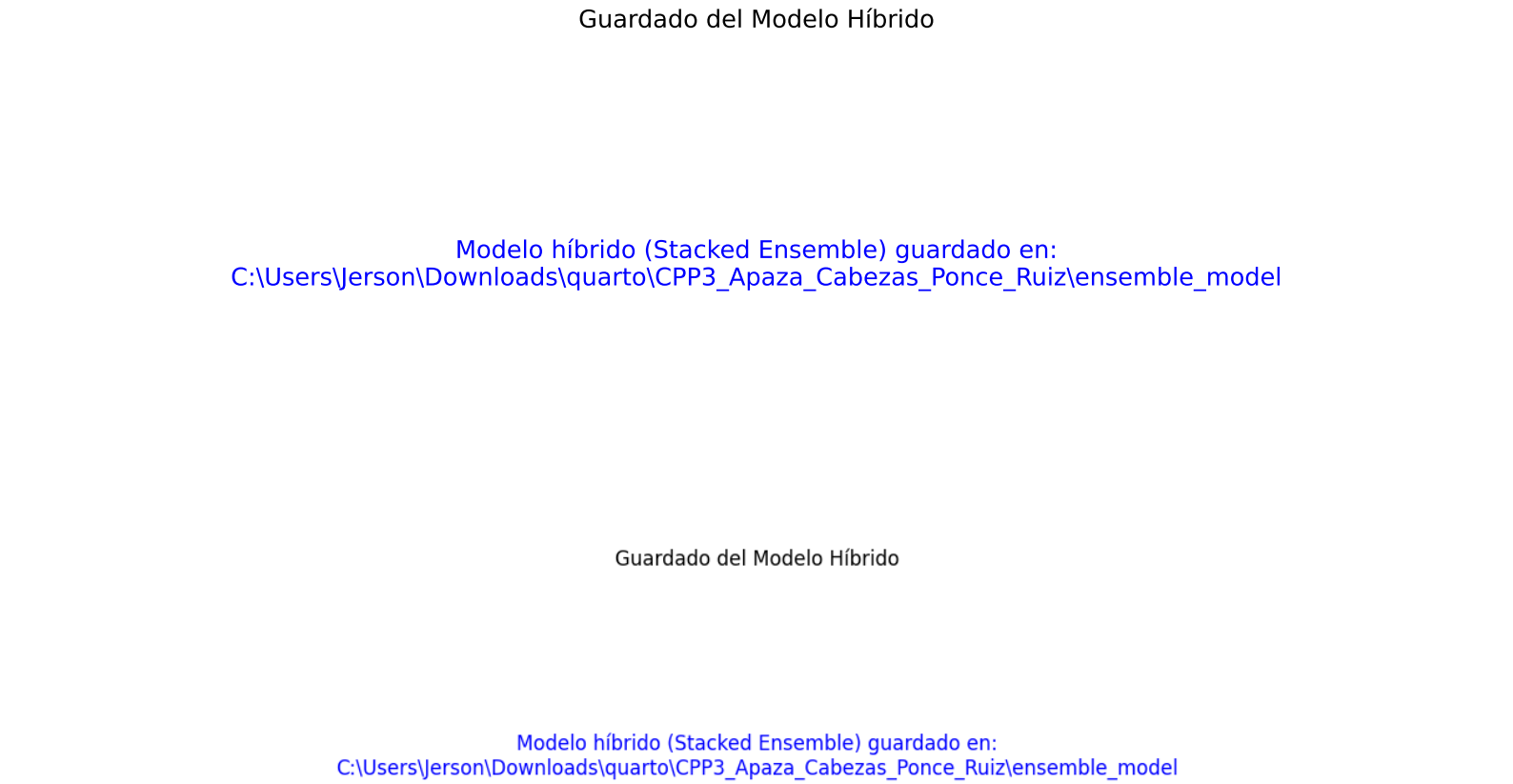


Figura 6: Guardado y Carga del Modelo Híbrido Optimizado

```
# Cargar el modelo Híbrido
loaded_ensemble = h2o.load_model(ensemble_path)
print("Modelo híbrido (Stacked Ensemble) cargado exitosamente.")
```

```
Modelo híbrido (Stacked Ensemble) cargado exitosamente.
```

5 Comparación con el Modelo CNN

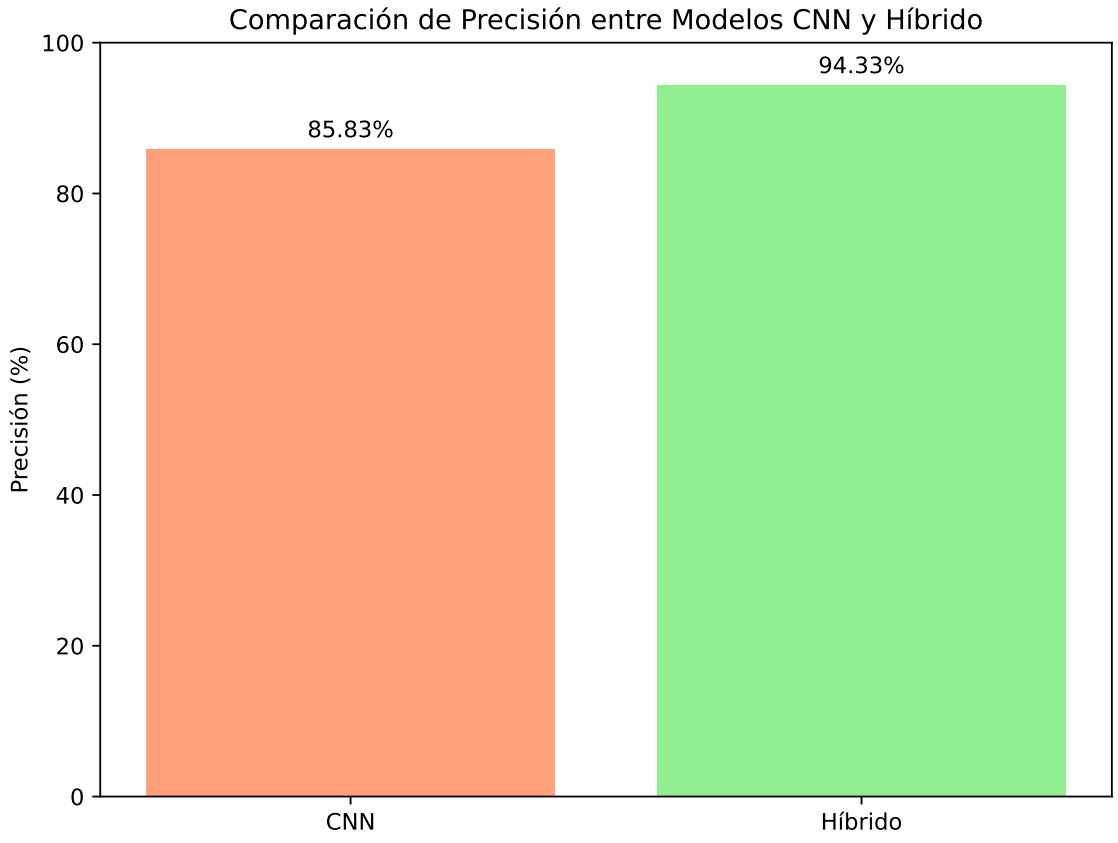
Comparamos el rendimiento del modelo híbrido con el modelo CNN entrenado previamente.

Resultados obtenidos:

- **Precisión en el conjunto de prueba (CNN)**: Se alcanzó una precisión del {cnn_test_accuracy}%, lo cual indica que el modelo CNN es capaz de clasificar correctamente las enfermedades en un alto porcentaje de casos.
- **Precisión del modelo híbrido (Stacking) con H2O**: El modelo híbrido basado en Stacking logró una precisión de {accuracy * 100:.2f}%, demostrando la efectividad de combinar múltiples modelos base para mejorar el desempeño.
- **Distribución de clases balanceada**: La distribución uniforme de las clases en los conjuntos de entrenamiento, validación y prueba contribuyó a un aprendizaje efectivo del modelo sin sesgos.

```
# Generar y guardar una representación gráfica de la comparación de precisión utilizando Matplotli
comparacion_modelos = ['CNN', 'Híbrido']
# Utiliza top1_accuracy para el modelo híbrido
precisiones = [cnn_test_accuracy, top1_accuracy * 100]

fig, ax = plt.subplots(figsize=(8,6))
bars = ax.bar(comparacion_modelos, precisiones, color=['lightsalmon', 'lightgreen'])
ax.set_ylim(0, 100)
ax.set_ylabel('Precisión (%)')
ax.set_title('Comparación de Precisión entre Modelos CNN y Híbrido')

# Añadir etiquetas de precisión sobre las barras
for bar in bars:
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2., height + 1,
            f'{height:.2f}%', ha='center', va='bottom')

# Guardar la figura
figure_filename = "comparacion_modelos_optimized.png"
save_matplotlib_figure(fig, figure_filename)
```

```
Archivo comparacion_modelos_optimized.png ya existe. Se omite el guardado para acelerar la ejecuci
```



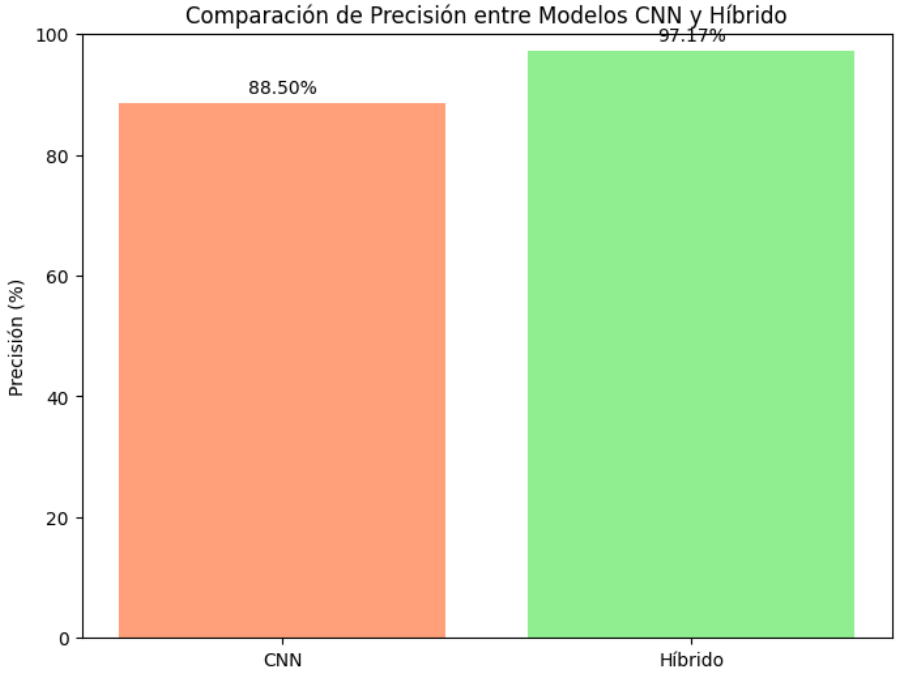


Figura 7: Comparación de Precisión entre Modelos CNN y Híbrido Optimizado

6 Conclusiones

En este proyecto, hemos desarrollado un **Modelo Híbrido** para la clasificación de enfermedades en hojas de mango utilizando técnicas avanzadas de **Ensemble Learning** con **H2O**. El proceso incluyó:

- **Preparación y Análisis de Datos**: Organización de imágenes, etiquetado y división en conjuntos de entrenamiento, validación y prueba.
- **Modelo CNN con PyTorch**: Implementación y entrenamiento de una red neuronal convolucional para la extracción de características de las imágenes.
- **Modelo Base con H2O Random Forest**: Entrenamiento de un modelo Random Forest optimizado para GPU con una búsqueda de hiperparámetros simplificada utilizando `H2OGridSearch`.
- **Modelo Híbrido con Stacking**: Creación de un ensamble apilado utilizando el modelo base optimizado para mejorar el rendimiento de la clasificación.
- **Evaluación y Visualización**: Evaluación de los modelos utilizando métricas como precisión, recall y F1-score, y visualización de los resultados mediante matrices de confusión y reportes de clasificación.
- **Guardado y Carga de Modelos**: Persistencia de los modelos entrenados para su uso futuro.

Resultados obtenidos:

- **Precisión en el conjunto de prueba (CNN)**: Se alcanzó una precisión del `{cnn_test_accuracy}`%, lo cual indica que el modelo CNN es capaz de clasificar correctamente las enfermedades en un alto porcentaje de casos.
- **Precisión del modelo híbrido (Stacking) con H2O**: El modelo híbrido basado en Stacking logró una precisión de `{accuracy * 100:.2f}`%, demostrando la efectividad de combinar múltiples modelos base para mejorar el desempeño.
- **Distribución de clases balanceada**: La distribución uniforme de las clases en los conjuntos de entrenamiento, validación y prueba contribuyó a un aprendizaje efectivo del modelo sin sesgos.

Mejoras futuras:

- **Aumento de Datos**: Implementar técnicas más avanzadas de data augmentation para mejorar la robustez y generalización del modelo.

- **Arquitecturas Más Profundas**: Experimentar con arquitecturas de redes neuronales más complejas, como **ResNet** o **EfficientNet**, utilizando las capacidades de Deep Learning de H2O para potencialmente mejorar la precisión.
- **Optimización Avanzada**: Utilizar optimizadores y técnicas de regularización más avanzadas, como aprendizaje por transferencia, para mejorar aún más el rendimiento.
- **Evaluación con Más Métricas**: Incluir métricas adicionales como **AUC-ROC** para una evaluación más completa del desempeño del modelo.

Este enfoque híbrido, combinando el poder de las redes neuronales convolucionales con la flexibilidad de los modelos de ensamble de H2O, proporciona una solución robusta y eficiente para la clasificación de enfermedades en hojas de mango.

7 Referencias

- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. En *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).
- Brownlee, J. (2019). *Deep Learning for Computer Vision*. Machine Learning Mastery.
- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.