IntroCV_Tarea_4_Sofía_Vits

June 8, 2025

Introducción a la Visión Computacional

Tarea 4

0.0.1 Magíster en Data Science

0.0.2 U. del Desarrollo

Estudiante: Sofía Vits

Profesor: Takeshi Asahi

Fecha de Entrega: Lunes 9, Junio 2025.

1 Introducción

En este trabajo se continuará con la clasificación de imágenes de papas sanas e infectadas por bacterias y hongos. Primero se pre-procesó el dataset "Potato Diseases Datasets" aplicando desenfoque Gaussiano, detección de bordes mediante la aplicación del filtro Sobel, aclarado de las imágenes mediante filtro Máximo, y oscurecimiento de imágenes al aplicar el filtro Mínimo.

Este dataset contiene 451 imágenes de papas clasificadas como infectadas por common scab, blackleg, dry rot, pink rot, y black scurf, papas sanas, y papas afectadas por otros tipos de infecciones. Considerando las enfermedades más comunes de la papa en Chile, se consideraron las clases common scab, blackleg, dry rot y papas sanas para realizar una clasificación multiclase mediante la aplicación del algoritmo. Esta muestra del dataset contiene 262 imágenes en total.

En la segunda parte de este trabajo se utilizó un algoritmo SVM multiclase, cuyos resultados serán mencionados para compararlos con los datos obtenidos al realizar fine-tuning del modelo de visión computacional ResNet50V2 y al aplicar el algoritmo AdaBoost utilizando árboles de decisión como estimador débil.

Se escogió el modelo ResNet50V2 debido a que esta red residual permite manejar de forma adecuada el problema del desvanecimiento de gradiente durante su entrenamiento, y porque clasifica de forma adecuada el dataset ImageNet.

Finalmente, se determinó el algoritmo más adecuado para la clasificación de la muestra del dataset de papas.

Fuente del dataset utilizado:

https://www.kaggle.com/datasets/mukaffimoin/potato-diseases-datasets/

F. T. J. Faria, M. Bin Moin, A. Al Wase, M. R. Sani, K. M. Hasib and M. S. Alam, "Classification of Potato Disease with Digital Image Processing Technique: A Hybrid Deep Learning Framework," 2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2023, pp. 0820-0826, doi: 10.1109/CCWC57344.2023.10099162.

2 Metodología

1.- Cargar la muestra del dataset a utilizar para los algoritmos de clasificación multiclase.

2.- Realizar fine-tuning de un modelo de visión computacional pre-entrenado, utilizando GPU para acelerar este proceso.

3.- Medir el tiempo mínimo, tiempo máximo, tiempo promedio y desviación estándard del tiempo de procesamiento del modelo de deep learning pre-entrenado.

4.- Implementar un algoritmo AdaBoost, midiendo sus métricas y tiempo de ejecución.

5.- Comparar los resultados del modelo de deep learning pre-entrenado y del algoritmo AdaBoost con los resultados obtenidos al utilizar el algoritmo SVM mediante métricas para verificar qué tipo de procesamiento permite obtener los mejores resultados.

3 Desarrollo

3.1 Propuesta de mejoras al sistema diseñado e implementado originalmente

Se propone aumentar la cantidad de imágenes del dataset, debido a que sólo contiene 262 muestras en total, lo cual es insuficiente para implementar algoritmos distintos a Support Vector Machines.

Se propone aplicar data augmentation para paliar la baja cantidad de imágenes del dataset.

También se sugiere utilizar arquitecturas de deep learning para crear los embeddings de las imágenes de papas.

Debido a que el algoritmo SVM consume mucha memoria RAM, es recomendable verificar si otros modelos de machine learning permiten obtener resultados similares utilizando una menor cantidad de memoria RAM.

3.2 Carga de librerías y definición de rutas de las carpetas que contienen las imágenes de la muestra del dataset de papas

[ ]:
```
from PIL import Image
from PIL import ImageDraw
from os.path import exists
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.ensemble import AdaBoostClassifier
from skimage.transform import resize
from skimage.io import imread
from sklearn.model_selection import train_test_split
from sklearn.metrics import␣
 ↪classification_report,accuracy_score,confusion_matrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import precision_score,recall_score,f1_score
import pandas as pd
import time
import seaborn as sns
import splitfolders
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Se cambió el nombre de directorio healthy_potato a potato, debido a que es posible que potato exista como clase en el dataset ImageNet.

[ ]:
```
os.chdir(r'/content/drive/MyDrive/potato/data/')

os.rename("healthy_potato","potato")

os.chdir(r'/content/drive/MyDrive/potato/data_g/')

os.rename("healthy_potato","potato")

os.chdir(r'/content/drive/MyDrive/potato/data_s/')

os.rename("healthy_potato","potato")

os.chdir(r'/content/drive/MyDrive/potato/data_max/')

os.rename("healthy_potato","potato")

os.chdir(r'/content/drive/MyDrive/potato/data_min/')

os.rename("healthy_potato","potato")
```

[ ]:
```
# Abrir una imagen de cada carpeta que contiene imágenes de papas enfermas o␣
 ↪sanas

inPath1  = '/content/drive/MyDrive/potato/data/common_scab'
inPath2  = '/content/drive/MyDrive/potato/data/blackleg'
inPath3  = '/content/drive/MyDrive/potato/data/dry_rot'
inPath4 = '/content/drive/MyDrive/potato/data/potato'
```

```
inPath5 = '/content/drive/MyDrive/potato/data_g/common_scab'
inPath6 = '/content/drive/MyDrive/potato/data_g/blackleg'
inPath7 = '/content/drive/MyDrive/potato/data_g/dry_rot'
inPath8 = '/content/drive/MyDrive/potato/data_g/potato'
inPath9 = '/content/drive/MyDrive/potato/data_s/common_scab'
inPath10 = '/content/drive/MyDrive/potato/data_s/blackleg'
inPath11 = '/content/drive/MyDrive/potato/data_s/dry_rot'
inPath12 = '/content/drive/MyDrive/potato/data_s/potato'
inPath13 = '/content/drive/MyDrive/potato/data_max/common_scab'
inPath14 = '/content/drive/MyDrive/potato/data_max/blackleg'
inPath15 = '/content/drive/MyDrive/potato/data_max/dry_rot'
inPath16 = '/content/drive/MyDrive/potato/data_max/potato'
inPath17 = '/content/drive/MyDrive/potato/data_min/common_scab'
inPath18 = '/content/drive/MyDrive/potato/data_min/blackleg'
inPath19 = '/content/drive/MyDrive/potato/data_min/dry_rot'
inPath20 = '/content/drive/MyDrive/potato/data_min/potato'
```

3.3 Resultado del pre-procesamiento del dataset de papas

En esta sección se muestran ejemplos de papas sanas e infectadas por blackleg, common scab, y dry rot. En todos los casos se comparan imágenes sin filtrar con fotografías a las cuales se les aplicó desenfoque Gaussiano, detección de bordes mediante un filtro Sobel, y filtros Máximo y Mínimo.

El filtro desenfoque Gaussiano fue aplicado a las imágenes del dataset para crear imágenes borrosas en escala de grises de las papas enfermas y sanas, utilizando para dicho propósito el parámetro sigma (desviación estándar).

El filtro Sobel detectó los bordes horizontales y verticales separadamente sobre las imágenes en escala de grises. El operador Sobel es de tipo diferencial.

El filtro Máximo seleccionó el mayor valor dentro de una ventana ordenada de valores de nivel de gris. Eliminó el ruido pimienta, aclarando la imagen original.

El filtro Mínimo seleccionó el menor valor dentro de una ventana ordenada de valores de nivel de gris. Eliminó el ruido sal, oscureciendo la imagen original.

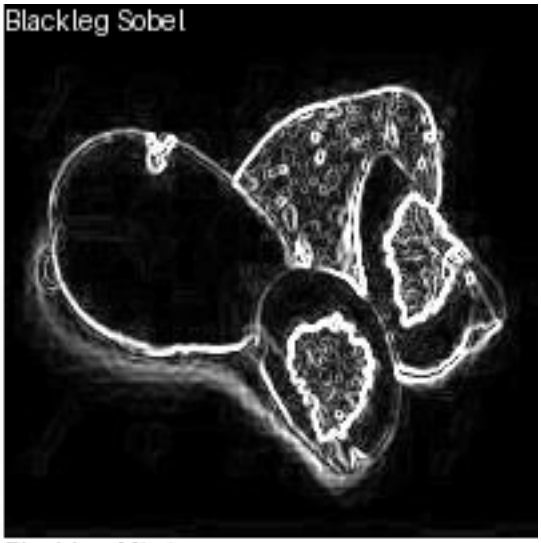
[ ]:
```
# Define grilla de imágenes
def image_grid(imgs, rows, cols):
    assert len(imgs) == rows*cols

    w, h = imgs[0].size
    grid = Image.new('RGB', size=(cols*w, rows*h))
    grid_w, grid_h = grid.size


    for i, img in enumerate(imgs):
        grid.paste(img, box=(i%cols*w, i//cols*h))
    return grid
```

[ ]:
```
# Compara papa afectada por blackleg antes y después de aplicar desenfoque␣
↪Gaussiano, filtros Sobel, Máximo y Mínimo

# Original
pil_im = Image.open('{}/20.jpg'.format(inPath2))

from PIL import ImageDraw
ImageDraw.Draw(pil_im).text((0, 0),'Blackleg', (0, 0, 0))

display(pil_im)

# Desenfoque Gaussiano
pil_im_g = Image.open('{}/20.jpg'.format(inPath6)).convert('RGB')
ImageDraw.Draw(pil_im_g).text((0, 0),'Blackleg Gaussian', (0, 0, 0))

# Sobel
pil_im_s = Image.open('{}/20.jpg'.format(inPath10)).convert('RGB')
ImageDraw.Draw(pil_im_s).text((0, 0),'Blackleg Sobel', (255, 255, 255))

# Máximo
pil_im_m = Image.open('{}/20.jpg'.format(inPath14)).convert('RGB')
ImageDraw.Draw(pil_im_m).text((0, 0),'Blackleg Maximum', (0, 0, 0))

# Mínimo
pil_im_mi = Image.open('{}/20.jpg'.format(inPath18)).convert('RGB')
ImageDraw.Draw(pil_im_mi).text((0, 0),'Blackleg Minimum', (0, 0, 0))

imgs = [pil_im_g, pil_im_s, pil_im_m, pil_im_mi]

grid = image_grid(imgs, rows=2, cols=2,)

display(grid)
```

Blackleg

Blackleg Gaussian

Blackleg Sobel



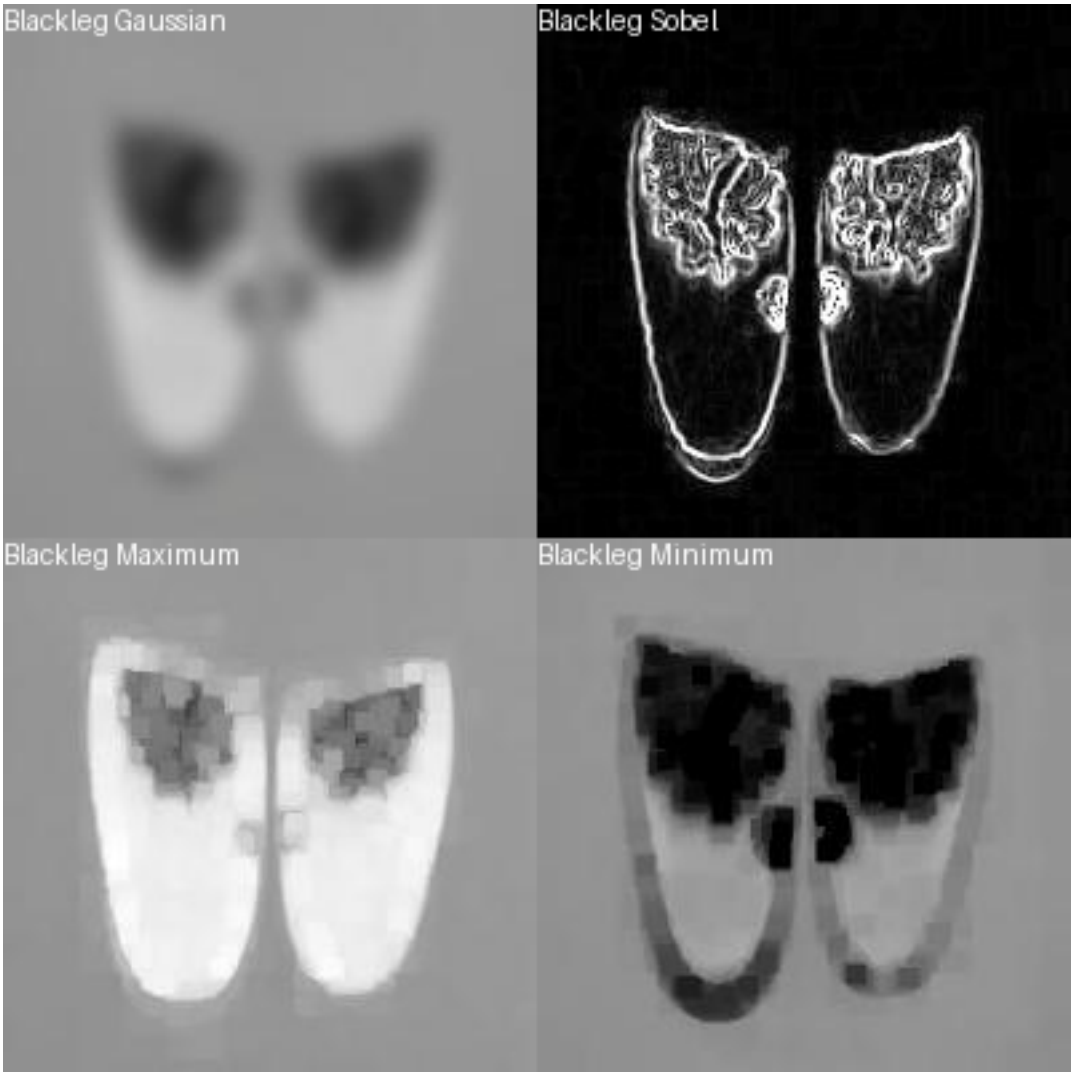
Blackleg Maximum

Blackleg Minimum

El ejemplo al cual se le aplicó el filtro Máximo no se distingue bien del fondo blanco. Este problema debiera dificultar el proceso de etiquetado realizado los algoritmos de deep learning y AdaBoost, dificultando su distinción de las otras clases de papas del dataset.

[ ]:
```
# Compara papa afectada por blackleg antes y después de aplicar desenfoque␣
↪Gaussiano, filtros Sobel, Máximo y Mínimo

# Original
pil_im2 = Image.open('{}/51.jpg'.format(inPath2))
ImageDraw.Draw(pil_im2).text((0, 0),'Blackleg', (255, 255, 255))
display(pil_im2)

# Desenfoque Gaussiano
pil_im_g2 = Image.open('{}/51.jpg'.format(inPath6)).convert('RGB')
ImageDraw.Draw(pil_im_g2).text((0, 0),'Blackleg Gaussian', (255, 255, 255))

# Sobel
pil_im_s2 = Image.open('{}/51.jpg'.format(inPath10)).convert('RGB')
ImageDraw.Draw(pil_im_s2).text((0, 0),'Blackleg Sobel', (255, 255, 255))

# Máximo
pil_im_m2 = Image.open('{}/51.jpg'.format(inPath14)).convert('RGB')
ImageDraw.Draw(pil_im_m2).text((0, 0),'Blackleg Maximum', (255, 255, 255))

# Mínimo
pil_im_mi2 = Image.open('{}/51.jpg'.format(inPath18)).convert('RGB')
ImageDraw.Draw(pil_im_mi2).text((0, 0),'Blackleg Minimum', (255, 255, 255))

imgs = [pil_im_g2, pil_im_s2, pil_im_m2, pil_im_mi2]

grid = image_grid(imgs, rows=2, cols=2)

display(grid)
```

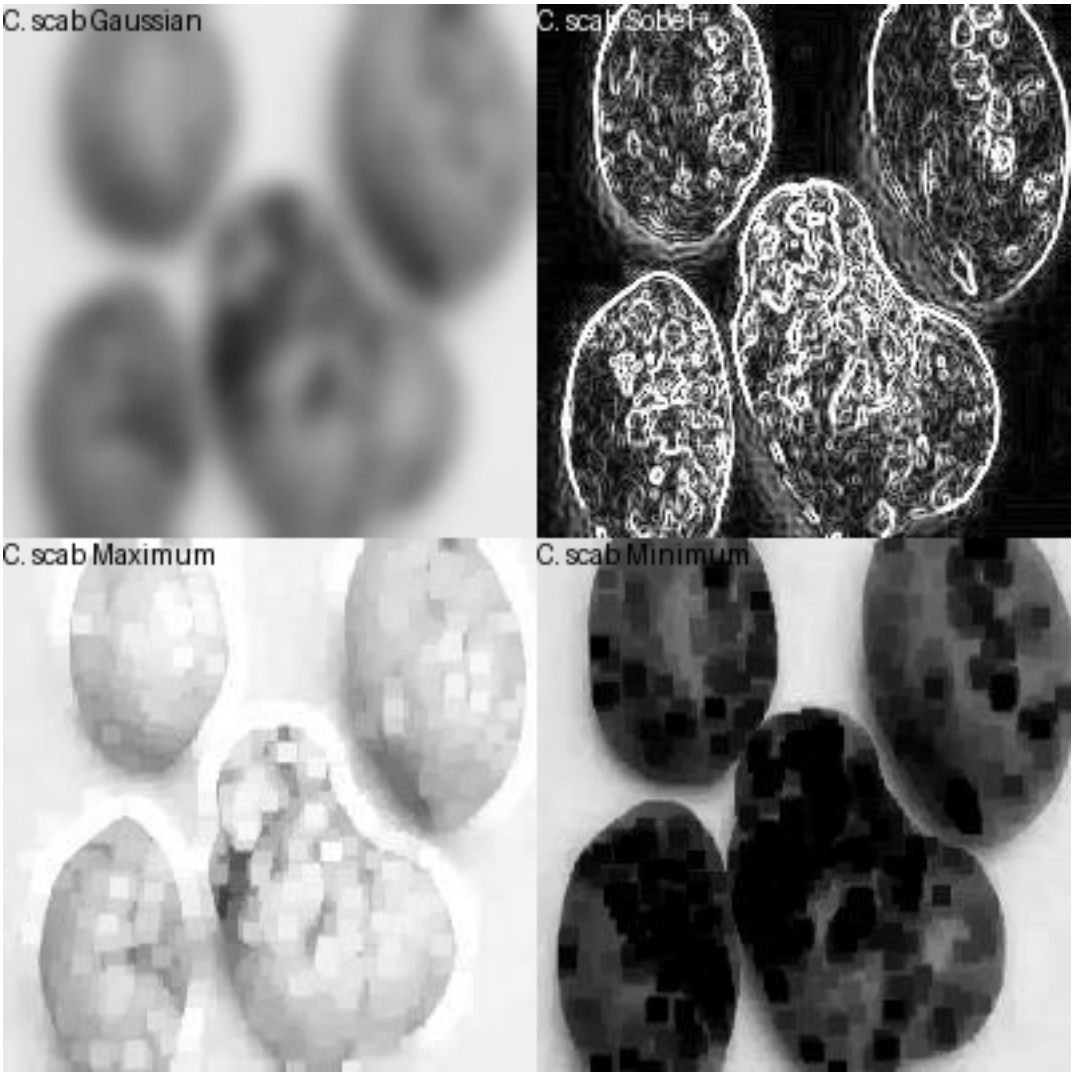

Blackleg


Blackleg Gaussian
Blackleg Sobel
Blackleg Maximum
Blackleg Minimum

Los bordes de la papa infectada con blackleg no se distinguen bien en el caso de la imagen a la cual se le aplicó un filtro Mínimo. Este problema debiera complicar la correcta clasificación de la imagen, confundiéndola con los otros tipos de papas enfermas.

[ ]:
```
# Compara papa afectada por common scab antes y después de aplicar desenfoque␣
 ↪Gaussiano, filtros Sobel, Máximo y Mínimo

# Original
pil_im3 = Image.open('{}/10.jpg'.format(inPath1))
ImageDraw.Draw(pil_im3).text((0, 0),'Common scab', (0, 0, 0))
display(pil_im3)

# Desenfoque Gaussiano
pil_im_g3 = Image.open('{}/10.jpg'.format(inPath5)).convert('RGB')
ImageDraw.Draw(pil_im_g3).text((0, 0),'C. scab Gaussian', (0, 0, 0))

# Sobel
pil_im_s3 = Image.open('{}/10.jpg'.format(inPath9)).convert('RGB')
ImageDraw.Draw(pil_im_s3).text((0, 0),'C. scab Sobel', (255, 255, 255))

# Máximo
pil_im_m3 = Image.open('{}/10.jpg'.format(inPath13)).convert('RGB')
ImageDraw.Draw(pil_im_m3).text((0, 0),'C. scab Maximum', (0, 0, 0))

# Mínimo
pil_im_mi3 = Image.open('{}/10.jpg'.format(inPath17)).convert('RGB')
ImageDraw.Draw(pil_im_mi3).text((0, 0),'C. scab Minimum', (0, 0, 0))

imgs = [pil_im_g3, pil_im_s3, pil_im_m3, pil_im_mi3]

grid = image_grid(imgs, rows=2, cols=2)

display(grid)
```

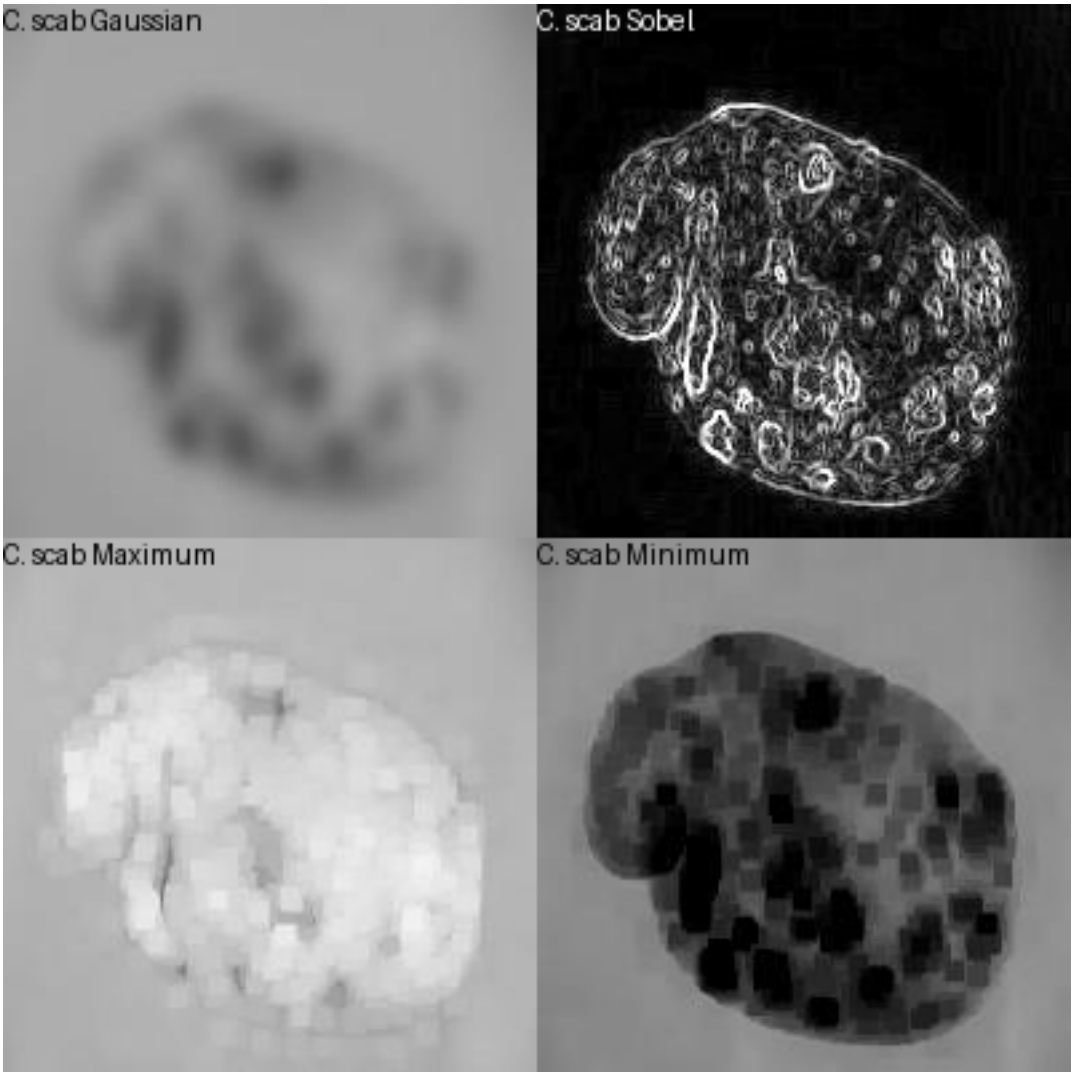

Common scab


C. scab Gaussian
C. scab Sobel
C. scab Maximum
C. scab Minimum

En este caso no se observan problemas de bordes difuminados al aplicar los filtros Máximo y Mínimo.

[ ]:
```
# Compara papa afectada por common scab antes y después de aplicar desenfoque␣
↪Gaussiano, filtros Sobel, Máximo y Mínimo

# Original
pil_im4 = Image.open('{}/31.jpg'.format(inPath1))
ImageDraw.Draw(pil_im4).text((0, 0),'Common scab', (0, 0, 0))
display(pil_im4)

# Desenfoque Gaussiano
pil_im_g4 = Image.open('{}/31.jpg'.format(inPath5)).convert('RGB')
ImageDraw.Draw(pil_im_g4).text((0, 0),'C. scab Gaussian', (0, 0, 0))

# Sobel
pil_im_s4 = Image.open('{}/31.jpg'.format(inPath9)).convert('RGB')
ImageDraw.Draw(pil_im_s4).text((0, 0),'C. scab Sobel', (255, 255, 255))

# Máximo
pil_im_m4 = Image.open('{}/31.jpg'.format(inPath13)).convert('RGB')
ImageDraw.Draw(pil_im_m4).text((0, 0),'C. scab Maximum', (0, 0, 0))

# Mínimo
pil_im_mi4 = Image.open('{}/31.jpg'.format(inPath17)).convert('RGB')
ImageDraw.Draw(pil_im_mi4).text((0, 0),'C. scab Minimum', (0, 0, 0))

imgs = [pil_im_g4, pil_im_s4, pil_im_m4, pil_im_mi4]

grid = image_grid(imgs, rows=2, cols=2)

display(grid)
```

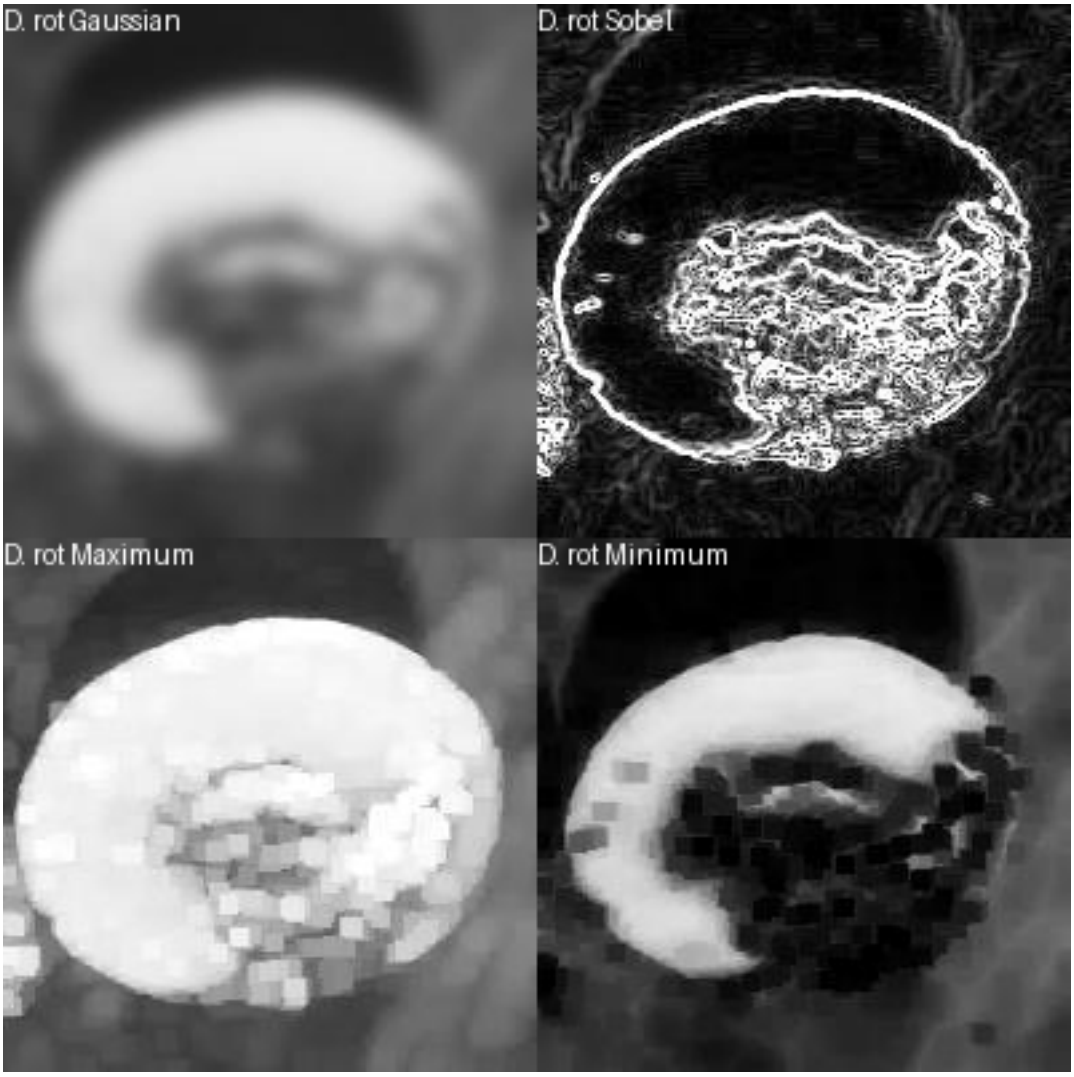

Common scab


C. scab Gaussian
C. scab Sobel
C. scab Maximum
C. scab Minimum

Los bordes de la imagen a la cual se le aplicó un filtro Máximo se difuminan con el fondo gris. Los algoritmos probablemente tampoco puedan distinguir bien esta papa con common scab de una papa que presente otra enfermedad dado que sus características han sido distorsionadas por el filtro.

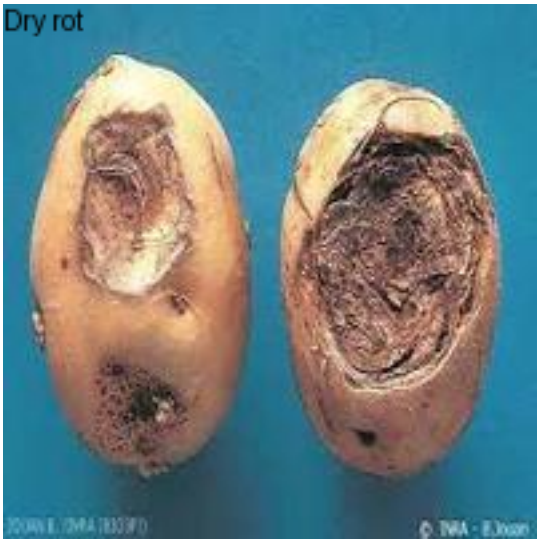
[ ]:
```
# Compara papa afectada por dry rot antes y después de aplicar desenfoque␣
↪Gaussiano, filtros Sobel, Máximo y Mínimo

# Original
pil_im5 = Image.open('{}/32.jpg'.format(inPath3))
ImageDraw.Draw(pil_im5).text((0, 0),'Dry rot', (255, 255, 255))
display(pil_im5)

# Desenfoque Gaussiano
pil_im_g5 = Image.open('{}/32.jpg'.format(inPath7)).convert('RGB')
ImageDraw.Draw(pil_im_g5).text((0, 0),'D. rot Gaussian', (255, 255, 255))

# Sobel
pil_im_s5 = Image.open('{}/32.jpg'.format(inPath11)).convert('RGB')
ImageDraw.Draw(pil_im_s5).text((0, 0),'D. rot Sobel', (255, 255, 255))

# Máximo
pil_im_m5 = Image.open('{}/32.jpg'.format(inPath15)).convert('RGB')
ImageDraw.Draw(pil_im_m5).text((0, 0),'D. rot Maximum', (255, 255, 255))

# Mínimo
pil_im_mi5 = Image.open('{}/32.jpg'.format(inPath19)).convert('RGB')
ImageDraw.Draw(pil_im_mi5).text((0, 0),'D. rot Minimum', (255, 255, 255))

imgs = [pil_im_g5, pil_im_s5, pil_im_m5, pil_im_mi5]

grid = image_grid(imgs, rows=2, cols=2)

display(grid)
```

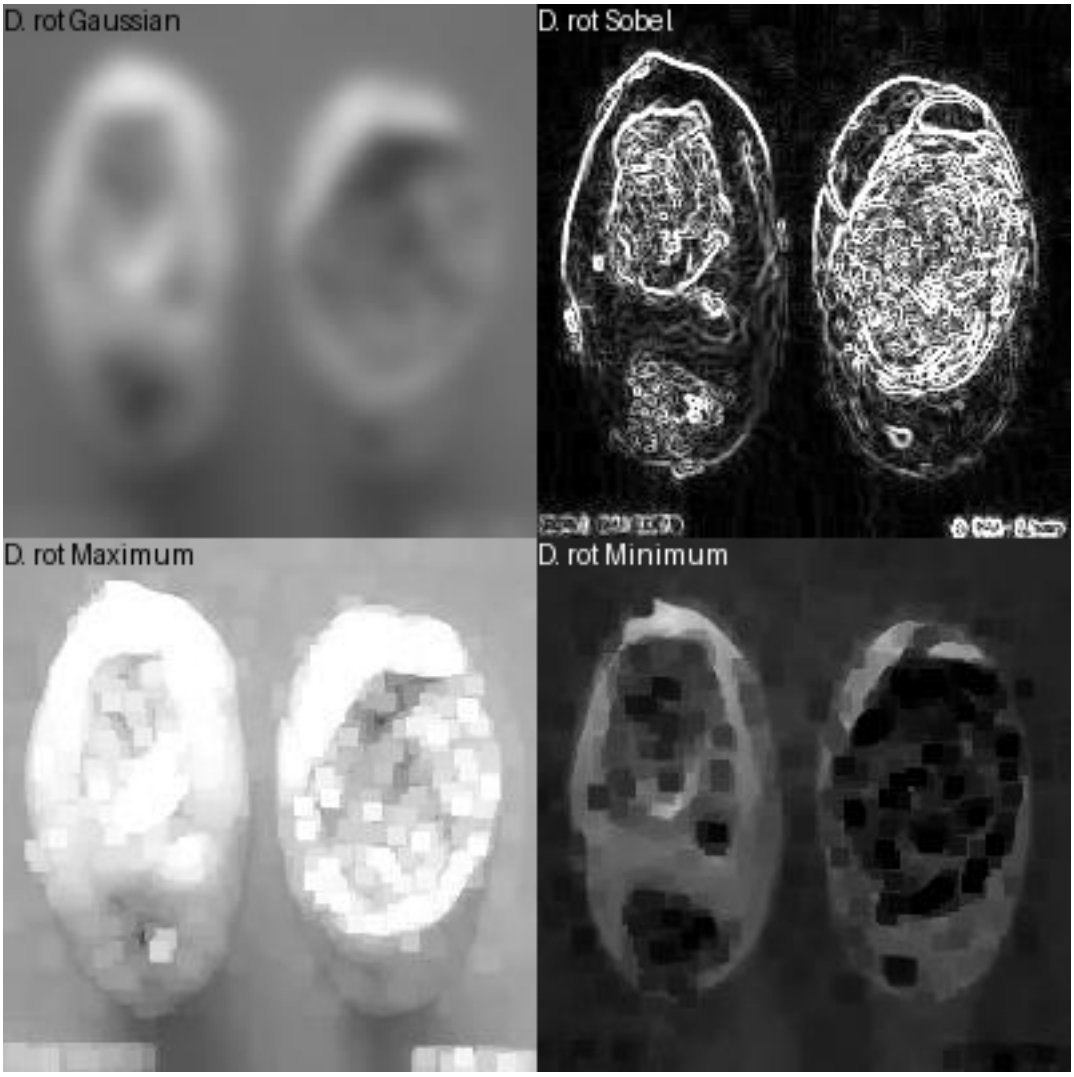

Dry rot


D. rot Gaussian
D. rot Sobel
D. rot Maximum
D. rot Minimum

Los bordes se diferencian adecuadamente del fondo en el caso de los filtros Máximo y Mínimo.

[ ]:
```
# Compara papa afectada por dry rot antes y después de aplicar desenfoque␣
↪Gaussiano, filtros Sobel, Máximo y Mínimo

# Original
pil_im6 = Image.open('{}/56.jpg'.format(inPath3)).convert('RGB')
ImageDraw.Draw(pil_im6).text((0, 0),'Dry rot', (0, 0, 0))
display(pil_im6)

# Desenfoque Gaussiano
pil_im_g6 = Image.open('{}/56.jpg'.format(inPath7)).convert('RGB')
ImageDraw.Draw(pil_im_g6).text((0, 0),'D. rot Gaussian', (0, 0, 0))

# Sobel
pil_im_s6 = Image.open('{}/56.jpg'.format(inPath11)).convert('RGB')
ImageDraw.Draw(pil_im_s6).text((0, 0),'D. rot Sobel', (255, 255, 255))

# Máximo
pil_im_m6 = Image.open('{}/56.jpg'.format(inPath15)).convert('RGB')
ImageDraw.Draw(pil_im_m6).text((0, 0),'D. rot Maximum', (0, 0, 0))

# Mínimo
pil_im_mi6 = Image.open('{}/56.jpg'.format(inPath19)).convert('RGB')
ImageDraw.Draw(pil_im_mi6).text((0, 0),'D. rot Minimum', (255, 255, 255))

imgs = [pil_im_g6, pil_im_s6, pil_im_m6, pil_im_mi6]

grid = image_grid(imgs, rows=2, cols=2)

display(grid)
```


Dry rot


D. rot Gaussian
D. rot Sobel
D. rot Maximum
D. rot Minimum

En el caso del filtro Mínimo parte de los bordes se pierden en el fondo de la imagen. Este problema debiera influir negativamente en su clasificación, pudiendo ser confundida por una papa con una enfermedad diferente.

[ ]:
```
# Compara papa sana antes y después de aplicar desenfoque Gaussiano, filtros␣
↪Sobel, Máximo y Mínimo

# Original
pil_im7 = Image.open('{}/55.jpg'.format(inPath4))
ImageDraw.Draw(pil_im7).text((0, 0),'Healthy', (0, 0, 0))
display(pil_im7)

# Desenfoque Gaussiano
pil_im_g7 = Image.open('{}/55.jpg'.format(inPath8)).convert('RGB')
ImageDraw.Draw(pil_im_g7).text((0, 0),'Healthy Gaussian', (0, 0, 0))

# Sobel
pil_im_s7 = Image.open('{}/55.jpg'.format(inPath12)).convert('RGB')
ImageDraw.Draw(pil_im_s7).text((0, 0),'Healthy Sobel', (255, 255, 255))

# Máximo
pil_im_m7 = Image.open('{}/55.jpg'.format(inPath16)).convert('RGB')
ImageDraw.Draw(pil_im_m7).text((0, 0),'Healthy Maximum', (0, 0, 0))

# Mínimo
pil_im_mi7 = Image.open('{}/55.jpg'.format(inPath20)).convert('RGB')
ImageDraw.Draw(pil_im_mi7).text((0, 0),'Healthy Minimum', (0, 0, 0))

imgs = [pil_im_g7, pil_im_s7, pil_im_m7, pil_im_mi7]

grid = image_grid(imgs, rows=2, cols=2)

display(grid)
```

Healthy

Healthy Gaussian

Healthy Sobel

Healthy Maximum

Healthy Minimum

Parte de los bordes de la imagen de las papas sanas no se distinguen bien del fondo en la fotografía a la cual se le aplicó filtro Máximo.

[ ]:
```
# Compara papa sana antes y después de aplicar desenfoque Gaussiano, filtros␣
 ↪Sobel, Máximo y Mínimo

# Original
pil_im8 = Image.open('{}/5.jpg'.format(inPath4))
ImageDraw.Draw(pil_im8).text((0, 0),'Healthy', (0, 0, 0))
display(pil_im8)

# Desenfoque Gaussiano
pil_im_g8 = Image.open('{}/5.jpg'.format(inPath8)).convert('RGB')
ImageDraw.Draw(pil_im_g8).text((0, 0),'Healthy Gaussian', (0, 0, 0))

# Sobel
pil_im_s8 = Image.open('{}/5.jpg'.format(inPath12)).convert('RGB')
ImageDraw.Draw(pil_im_s8).text((0, 0),'Healthy Sobel', (255, 255, 255))

# Máximo
pil_im_m8 = Image.open('{}/5.jpg'.format(inPath16)).convert('RGB')
ImageDraw.Draw(pil_im_m8).text((0, 0),'Healthy Maximum', (0, 0, 0))

# Mínimo
pil_im_mi8 = Image.open('{}/5.jpg'.format(inPath20)).convert('RGB')
ImageDraw.Draw(pil_im_mi8).text((0, 0),'Healthy Minimum', (0, 0, 0))

imgs = [pil_im_g8, pil_im_s8, pil_im_m8, pil_im_mi8]

grid = image_grid(imgs, rows=2, cols=2)

display(grid)
```

Healthy

Healthy Gaussian

Healthy Sobel

Healthy Maximum

Healthy Minimum

Se pierde gran parte de la papa en el fondo blanco en la fotografía a la cual se le aplicó el filtro Máximo. Posiblemente los algoritmos de clasificación no puedan distinguirla correctamente de las papas enfermas presentes en el dataset.

3.4 Fine-tuning modelo ResNet50V2 para dataset sin filtrar

Solamente se realizó una vez el paso de dividir las carpetas en train, validation y test. En pruebas posteriores se comentó esa parte del código debido a que las carpetas fueron almacenadas en Google Drive.

[ ]:
```
# Dividir carpeta en train, validation y test

path_model = "/content/drive/MyDrive/potato/data_model/" # Carpeta para guardar␣
 ↪imágenes
# Crea la carpeta de salida si no existe
if not os.path.exists(path_model):
    os.makedirs(path_model)

splitfolders.ratio("/content/drive/MyDrive/potato/data/", output="/content/
 ↪drive/MyDrive/potato/data_model/",
    seed=42, ratio=(.8, .1, .1), group_prefix=None, move=False) # default values
```

[ ]:
```
train_dir = os.path.join(path_model, 'train')
validation_dir = os.path.join(path_model, 'val')
test_dir = os.path.join(path_model,'test')

BATCH_SIZE = 9
IMG_SIZE = (200, 200)

train_dataset = tf.keras.utils.image_dataset_from_directory(train_dir,
                                                            shuffle=True,
                                                           ␣
 ↪batch_size=BATCH_SIZE,
                                                            image_size=IMG_SIZE)
```

```
Found 209 files belonging to 4 classes.
```

[ ]:
```
validation_dataset = tf.keras.utils.image_dataset_from_directory(validation_dir,
                                                                 shuffle=True,
                                                                ␣
 ↪batch_size=BATCH_SIZE,
                                                                ␣
 ↪image_size=IMG_SIZE)
```

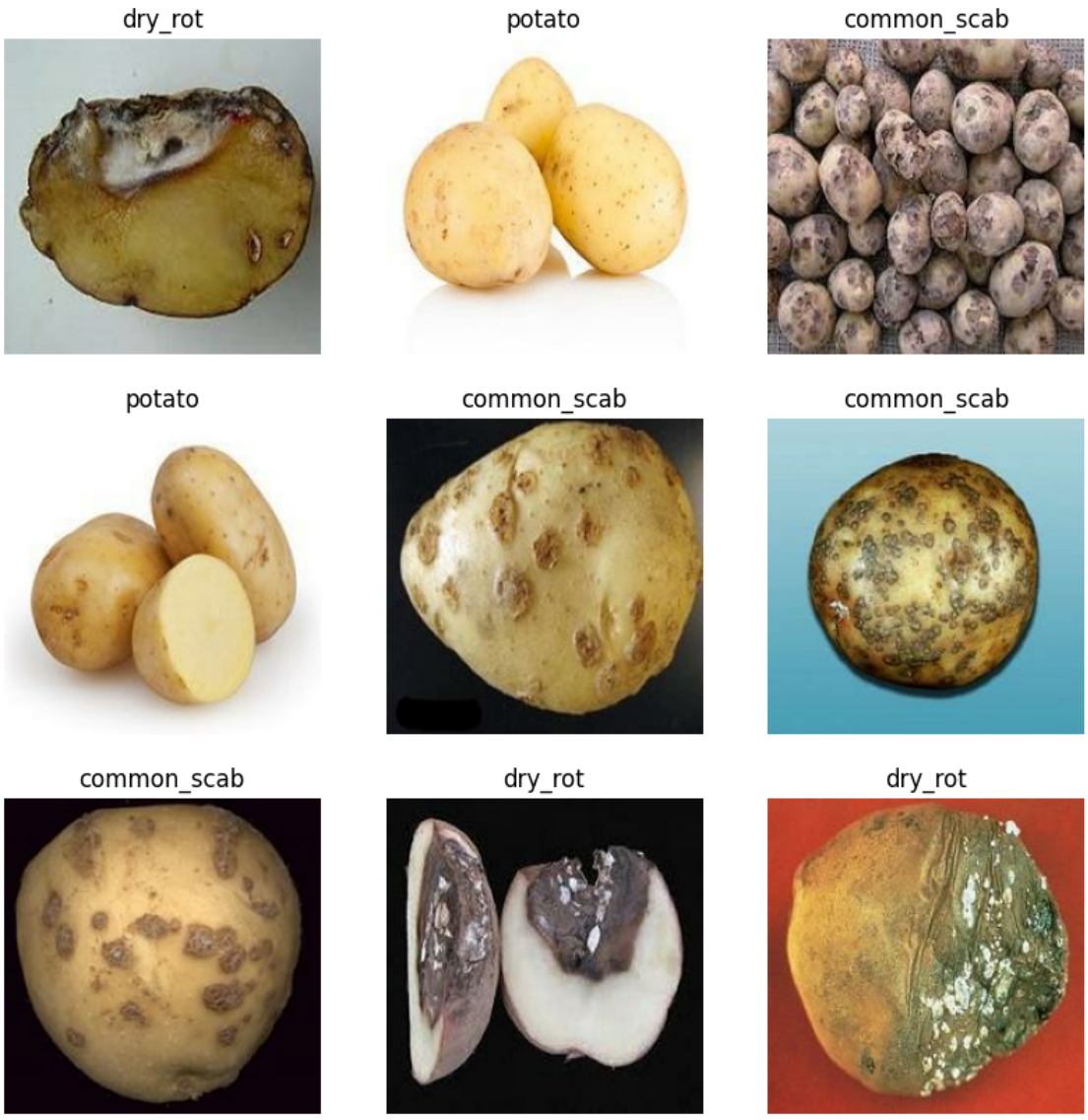
```
Found 26 files belonging to 4 classes.
```

[ ]:
```
test_dataset = tf.keras.utils.image_dataset_from_directory(test_dir,
                                                           shuffle=True,
                                                           ␣
 ↪batch_size=BATCH_SIZE,
                                                           ␣
 ↪image_size=IMG_SIZE)
```

```
Found 27 files belonging to 4 classes.
```

Se visualiza una muestra de las imágenes del conjunto de entrenamiento y sus respectivas etiquetas en una cuadrícula de 3x3.

[ ]:
```
class_names = train_dataset.class_names

plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```



[ ]:
```
# Ajusta parametros de memoria automaticamente a la capacidad del computador
AUTOTUNE = tf.data.AUTOTUNE

train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

[ ]:
```
# Aplica data augmentation al dataset de papas sin filtrar

data_augmentation = tf.keras.Sequential([
  tf.keras.layers.RandomFlip('horizontal'),
  tf.keras.layers.RandomRotation(0.1),
```

```
  tf.keras.layers.RandomTranslation(height_factor=0.1, width_factor=0.1),
  tf.keras.layers.RandomContrast(factor=0.1),
])
```

[ ]:
```
# Define el modelo que será aplicado para transformar las imágenes en vectores
preprocess_input = tf.keras.applications.resnet_v2.preprocess_input
```

[ ]:
```
# Crea el modelo base a partir del modelo pre-entrenado ResNet50V2
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tf.keras.applications.ResNet50V2(input_shape=IMG_SHAPE,
                                              include_top=False,
                                              weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/resnet50v2_weights_tf_dim_ordering_tf_kernels_notop.h5
94668760/94668760              3s
0us/step
```

En este paso se convierte a las imágenes en bloques de características de tamaño 7x7x2048.

[ ]:
```
image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
print(feature_batch.shape)
```

```
(9, 7, 7, 2048)
```

Se congela la base convolucional para evitar que sus pesos se actualicen durante el entrenamiento.

[ ]:
```
base_model.trainable = False
```

[ ]:
```
# Arquitectura del modelo base ResNet50V2
base_model.summary()
```

```
Model: "resnet50v2"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 200, 200, 3)	0	-
conv1_pad (ZeroPadding2D)	(None, 206, 206, 3)	0	input_layer_1[0]…
conv1_conv (Conv2D)	(None, 100, 100, 64)	9,472	conv1_pad[0][0]
pool1_pad (ZeroPadding2D)	(None, 102, 102, 64)	0	conv1_conv[0][0]

```
pool1_pool              (None, 50, 50,              0   pool1_pad[0][0]
(MaxPooling2D)          64)

conv2_block1_preac…     (None, 50, 50,            256   pool1_pool[0][0]
(BatchNormalizatio…     64)

conv2_block1_preac…     (None, 50, 50,              0   conv2_block1_pre…
(Activation)            64)

conv2_block1_1_conv     (None, 50, 50,          4,096   conv2_block1_pre…
(Conv2D)                64)

conv2_block1_1_bn       (None, 50, 50,            256   conv2_block1_1_c…
(BatchNormalizatio…     64)

conv2_block1_1_relu     (None, 50, 50,              0   conv2_block1_1_b…
(Activation)            64)

conv2_block1_2_pad      (None, 52, 52,              0   conv2_block1_1_r…
(ZeroPadding2D)         64)

conv2_block1_2_conv     (None, 50, 50,         36,864   conv2_block1_2_p…
(Conv2D)                64)

conv2_block1_2_bn       (None, 50, 50,            256   conv2_block1_2_c…
(BatchNormalizatio…     64)

conv2_block1_2_relu     (None, 50, 50,              0   conv2_block1_2_b…
(Activation)            64)

conv2_block1_0_conv     (None, 50, 50,         16,640   conv2_block1_pre…
(Conv2D)                256)

conv2_block1_3_conv     (None, 50, 50,         16,640   conv2_block1_2_r…
(Conv2D)                256)

conv2_block1_out        (None, 50, 50,              0   conv2_block1_0_c…
(Add)                   256)                            conv2_block1_3_c…

conv2_block2_preac…     (None, 50, 50,          1,024   conv2_block1_out…
(BatchNormalizatio…     256)

conv2_block2_preac…     (None, 50, 50,              0   conv2_block2_pre…
(Activation)            256)

conv2_block2_1_conv     (None, 50, 50,         16,384   conv2_block2_pre…
(Conv2D)                64)
```

```
conv2_block2_1_bn     (None, 50, 50,            256   conv2_block2_1_c…
(BatchNormalizatio…   64)

conv2_block2_1_relu   (None, 50, 50,              0   conv2_block2_1_b…
(Activation)          64)

conv2_block2_2_pad    (None, 52, 52,              0   conv2_block2_1_r…
(ZeroPadding2D)       64)

conv2_block2_2_conv   (None, 50, 50,         36,864   conv2_block2_2_p…
(Conv2D)              64)

conv2_block2_2_bn     (None, 50, 50,            256   conv2_block2_2_c…
(BatchNormalizatio…   64)

conv2_block2_2_relu   (None, 50, 50,              0   conv2_block2_2_b…
(Activation)          64)

conv2_block2_3_conv   (None, 50, 50,         16,640   conv2_block2_2_r…
(Conv2D)              256)

conv2_block2_out      (None, 50, 50,              0   conv2_block1_out…
(Add)                 256)                          conv2_block2_3_c…

conv2_block3_preac…   (None, 50, 50,          1,024   conv2_block2_out…
(BatchNormalizatio…   256)

conv2_block3_preac…   (None, 50, 50,              0   conv2_block3_pre…
(Activation)          256)

conv2_block3_1_conv   (None, 50, 50,         16,384   conv2_block3_pre…
(Conv2D)              64)

conv2_block3_1_bn     (None, 50, 50,            256   conv2_block3_1_c…
(BatchNormalizatio…   64)

conv2_block3_1_relu   (None, 50, 50,              0   conv2_block3_1_b…
(Activation)          64)

conv2_block3_2_pad    (None, 52, 52,              0   conv2_block3_1_r…
(ZeroPadding2D)       64)

conv2_block3_2_conv   (None, 25, 25,         36,864   conv2_block3_2_p…
(Conv2D)              64)

conv2_block3_2_bn     (None, 25, 25,            256   conv2_block3_2_c…
(BatchNormalizatio…   64)
```

```
conv2_block3_2_relu   (None, 25, 25,              0   conv2_block3_2_b…
(Activation)          64)

max_pooling2d         (None, 25, 25,              0   conv2_block2_out…
(MaxPooling2D)        256)

conv2_block3_3_conv   (None, 25, 25,          16,640   conv2_block3_2_r…
(Conv2D)              256)

conv2_block3_out      (None, 25, 25,              0   max_pooling2d[0]…
(Add)                 256)                            conv2_block3_3_c…

conv3_block1_preac…   (None, 25, 25,           1,024   conv2_block3_out…
(BatchNormalizatio…   256)

conv3_block1_preac…   (None, 25, 25,              0   conv3_block1_pre…
(Activation)          256)

conv3_block1_1_conv   (None, 25, 25,          32,768   conv3_block1_pre…
(Conv2D)              128)

conv3_block1_1_bn     (None, 25, 25,             512   conv3_block1_1_c…
(BatchNormalizatio…   128)

conv3_block1_1_relu   (None, 25, 25,              0   conv3_block1_1_b…
(Activation)          128)

conv3_block1_2_pad    (None, 27, 27,              0   conv3_block1_1_r…
(ZeroPadding2D)       128)

conv3_block1_2_conv   (None, 25, 25,         147,456   conv3_block1_2_p…
(Conv2D)              128)

conv3_block1_2_bn     (None, 25, 25,             512   conv3_block1_2_c…
(BatchNormalizatio…   128)

conv3_block1_2_relu   (None, 25, 25,              0   conv3_block1_2_b…
(Activation)          128)

conv3_block1_0_conv   (None, 25, 25,         131,584   conv3_block1_pre…
(Conv2D)              512)

conv3_block1_3_conv   (None, 25, 25,          66,048   conv3_block1_2_r…
(Conv2D)              512)

conv3_block1_out      (None, 25, 25,              0   conv3_block1_0_c…
(Add)                 512)                            conv3_block1_3_c…
```

conv3_block2_preac… (BatchNormalizatio…	(None, 25, 25, 512)	2,048	conv3_block1_out…
conv3_block2_preac… (Activation)	(None, 25, 25, 512)	0	conv3_block2_pre…
conv3_block2_1_conv (Conv2D)	(None, 25, 25, 128)	65,536	conv3_block2_pre…
conv3_block2_1_bn (BatchNormalizatio…	(None, 25, 25, 128)	512	conv3_block2_1_c…
conv3_block2_1_relu (Activation)	(None, 25, 25, 128)	0	conv3_block2_1_b…
conv3_block2_2_pad (ZeroPadding2D)	(None, 27, 27, 128)	0	conv3_block2_1_r…
conv3_block2_2_conv (Conv2D)	(None, 25, 25, 128)	147,456	conv3_block2_2_p…
conv3_block2_2_bn (BatchNormalizatio…	(None, 25, 25, 128)	512	conv3_block2_2_c…
conv3_block2_2_relu (Activation)	(None, 25, 25, 128)	0	conv3_block2_2_b…
conv3_block2_3_conv (Conv2D)	(None, 25, 25, 512)	66,048	conv3_block2_2_r…
conv3_block2_out (Add)	(None, 25, 25, 512)	0	conv3_block1_out… conv3_block2_3_c…
conv3_block3_preac… (BatchNormalizatio…	(None, 25, 25, 512)	2,048	conv3_block2_out…
conv3_block3_preac… (Activation)	(None, 25, 25, 512)	0	conv3_block3_pre…
conv3_block3_1_conv (Conv2D)	(None, 25, 25, 128)	65,536	conv3_block3_pre…
conv3_block3_1_bn (BatchNormalizatio…	(None, 25, 25, 128)	512	conv3_block3_1_c…
conv3_block3_1_relu (Activation)	(None, 25, 25, 128)	0	conv3_block3_1_b…

```
conv3_block3_2_pad    (None, 27, 27,              0   conv3_block3_1_r…
(ZeroPadding2D)       128)

conv3_block3_2_conv   (None, 25, 25,        147,456   conv3_block3_2_p…
(Conv2D)              128)

conv3_block3_2_bn     (None, 25, 25,            512   conv3_block3_2_c…
(BatchNormalizatio…   128)

conv3_block3_2_relu   (None, 25, 25,              0   conv3_block3_2_b…
(Activation)          128)

conv3_block3_3_conv   (None, 25, 25,         66,048   conv3_block3_2_r…
(Conv2D)              512)

conv3_block3_out      (None, 25, 25,              0   conv3_block2_out…
(Add)                 512)                          conv3_block3_3_c…

conv3_block4_preac…   (None, 25, 25,          2,048   conv3_block3_out…
(BatchNormalizatio…   512)

conv3_block4_preac…   (None, 25, 25,              0   conv3_block4_pre…
(Activation)          512)

conv3_block4_1_conv   (None, 25, 25,         65,536   conv3_block4_pre…
(Conv2D)              128)

conv3_block4_1_bn     (None, 25, 25,            512   conv3_block4_1_c…
(BatchNormalizatio…   128)

conv3_block4_1_relu   (None, 25, 25,              0   conv3_block4_1_b…
(Activation)          128)

conv3_block4_2_pad    (None, 27, 27,              0   conv3_block4_1_r…
(ZeroPadding2D)       128)

conv3_block4_2_conv   (None, 13, 13,        147,456   conv3_block4_2_p…
(Conv2D)              128)

conv3_block4_2_bn     (None, 13, 13,            512   conv3_block4_2_c…
(BatchNormalizatio…   128)

conv3_block4_2_relu   (None, 13, 13,              0   conv3_block4_2_b…
(Activation)          128)

max_pooling2d_1       (None, 13, 13,              0   conv3_block3_out…
(MaxPooling2D)        512)
```

```
conv3_block4_3_conv   (None, 13, 13,          66,048   conv3_block4_2_r…
(Conv2D)              512)

conv3_block4_out      (None, 13, 13,              0    max_pooling2d_1[…
(Add)                 512)                             conv3_block4_3_c…

conv4_block1_preac…   (None, 13, 13,           2,048   conv3_block4_out…
(BatchNormalizatio…   512)

conv4_block1_preac…   (None, 13, 13,              0    conv4_block1_pre…
(Activation)          512)

conv4_block1_1_conv   (None, 13, 13,         131,072   conv4_block1_pre…
(Conv2D)              256)

conv4_block1_1_bn     (None, 13, 13,           1,024   conv4_block1_1_c…
(BatchNormalizatio…   256)

conv4_block1_1_relu   (None, 13, 13,              0    conv4_block1_1_b…
(Activation)          256)

conv4_block1_2_pad    (None, 15, 15,              0    conv4_block1_1_r…
(ZeroPadding2D)       256)

conv4_block1_2_conv   (None, 13, 13,         589,824   conv4_block1_2_p…
(Conv2D)              256)

conv4_block1_2_bn     (None, 13, 13,           1,024   conv4_block1_2_c…
(BatchNormalizatio…   256)

conv4_block1_2_relu   (None, 13, 13,              0    conv4_block1_2_b…
(Activation)          256)

conv4_block1_0_conv   (None, 13, 13,         525,312   conv4_block1_pre…
(Conv2D)              1024)

conv4_block1_3_conv   (None, 13, 13,         263,168   conv4_block1_2_r…
(Conv2D)              1024)

conv4_block1_out      (None, 13, 13,              0    conv4_block1_0_c…
(Add)                 1024)                            conv4_block1_3_c…

conv4_block2_preac…   (None, 13, 13,           4,096   conv4_block1_out…
(BatchNormalizatio…   1024)

conv4_block2_preac…   (None, 13, 13,              0    conv4_block2_pre…
(Activation)          1024)
```

```
conv4_block2_1_conv   (None, 13, 13,        262,144   conv4_block2_pre…
(Conv2D)              256)

conv4_block2_1_bn     (None, 13, 13,          1,024   conv4_block2_1_c…
(BatchNormalizatio…   256)

conv4_block2_1_relu   (None, 13, 13,              0   conv4_block2_1_b…
(Activation)          256)

conv4_block2_2_pad    (None, 15, 15,              0   conv4_block2_1_r…
(ZeroPadding2D)       256)

conv4_block2_2_conv   (None, 13, 13,        589,824   conv4_block2_2_p…
(Conv2D)              256)

conv4_block2_2_bn     (None, 13, 13,          1,024   conv4_block2_2_c…
(BatchNormalizatio…   256)

conv4_block2_2_relu   (None, 13, 13,              0   conv4_block2_2_b…
(Activation)          256)

conv4_block2_3_conv   (None, 13, 13,        263,168   conv4_block2_2_r…
(Conv2D)              1024)

conv4_block2_out      (None, 13, 13,              0   conv4_block1_out…
(Add)                 1024)                         conv4_block2_3_c…

conv4_block3_preac…   (None, 13, 13,          4,096   conv4_block2_out…
(BatchNormalizatio…   1024)

conv4_block3_preac…   (None, 13, 13,              0   conv4_block3_pre…
(Activation)          1024)

conv4_block3_1_conv   (None, 13, 13,        262,144   conv4_block3_pre…
(Conv2D)              256)

conv4_block3_1_bn     (None, 13, 13,          1,024   conv4_block3_1_c…
(BatchNormalizatio…   256)

conv4_block3_1_relu   (None, 13, 13,              0   conv4_block3_1_b…
(Activation)          256)

conv4_block3_2_pad    (None, 15, 15,              0   conv4_block3_1_r…
(ZeroPadding2D)       256)

conv4_block3_2_conv   (None, 13, 13,        589,824   conv4_block3_2_p…
(Conv2D)              256)
```

```
conv4_block3_2_bn     (None, 13, 13,          1,024   conv4_block3_2_c…
(BatchNormalizatio…   256)

conv4_block3_2_relu   (None, 13, 13,              0   conv4_block3_2_b…
(Activation)          256)

conv4_block3_3_conv   (None, 13, 13,        263,168   conv4_block3_2_r…
(Conv2D)              1024)

conv4_block3_out      (None, 13, 13,              0   conv4_block2_out…
(Add)                 1024)                           conv4_block3_3_c…

conv4_block4_preac…   (None, 13, 13,          4,096   conv4_block3_out…
(BatchNormalizatio…   1024)

conv4_block4_preac…   (None, 13, 13,              0   conv4_block4_pre…
(Activation)          1024)

conv4_block4_1_conv   (None, 13, 13,        262,144   conv4_block4_pre…
(Conv2D)              256)

conv4_block4_1_bn     (None, 13, 13,          1,024   conv4_block4_1_c…
(BatchNormalizatio…   256)

conv4_block4_1_relu   (None, 13, 13,              0   conv4_block4_1_b…
(Activation)          256)

conv4_block4_2_pad    (None, 15, 15,              0   conv4_block4_1_r…
(ZeroPadding2D)       256)

conv4_block4_2_conv   (None, 13, 13,        589,824   conv4_block4_2_p…
(Conv2D)              256)

conv4_block4_2_bn     (None, 13, 13,          1,024   conv4_block4_2_c…
(BatchNormalizatio…   256)

conv4_block4_2_relu   (None, 13, 13,              0   conv4_block4_2_b…
(Activation)          256)

conv4_block4_3_conv   (None, 13, 13,        263,168   conv4_block4_2_r…
(Conv2D)              1024)

conv4_block4_out      (None, 13, 13,              0   conv4_block3_out…
(Add)                 1024)                           conv4_block4_3_c…

conv4_block5_preac…   (None, 13, 13,          4,096   conv4_block4_out…
(BatchNormalizatio…   1024)
```

conv4_block5_preac… (Activation)	(None, 13, 13, 1024)	0	conv4_block5_pre…
conv4_block5_1_conv (Conv2D)	(None, 13, 13, 256)	262,144	conv4_block5_pre…
conv4_block5_1_bn (BatchNormalizatio…	(None, 13, 13, 256)	1,024	conv4_block5_1_c…
conv4_block5_1_relu (Activation)	(None, 13, 13, 256)	0	conv4_block5_1_b…
conv4_block5_2_pad (ZeroPadding2D)	(None, 15, 15, 256)	0	conv4_block5_1_r…
conv4_block5_2_conv (Conv2D)	(None, 13, 13, 256)	589,824	conv4_block5_2_p…
conv4_block5_2_bn (BatchNormalizatio…	(None, 13, 13, 256)	1,024	conv4_block5_2_c…
conv4_block5_2_relu (Activation)	(None, 13, 13, 256)	0	conv4_block5_2_b…
conv4_block5_3_conv (Conv2D)	(None, 13, 13, 1024)	263,168	conv4_block5_2_r…
conv4_block5_out (Add)	(None, 13, 13, 1024)	0	conv4_block4_out… conv4_block5_3_c…
conv4_block6_preac… (BatchNormalizatio…	(None, 13, 13, 1024)	4,096	conv4_block5_out…
conv4_block6_preac… (Activation)	(None, 13, 13, 1024)	0	conv4_block6_pre…
conv4_block6_1_conv (Conv2D)	(None, 13, 13, 256)	262,144	conv4_block6_pre…
conv4_block6_1_bn (BatchNormalizatio…	(None, 13, 13, 256)	1,024	conv4_block6_1_c…
conv4_block6_1_relu (Activation)	(None, 13, 13, 256)	0	conv4_block6_1_b…
conv4_block6_2_pad (ZeroPadding2D)	(None, 15, 15, 256)	0	conv4_block6_1_r…

```
conv4_block6_2_conv   (None, 7, 7, 256)      589,824   conv4_block6_2_p…
(Conv2D)

conv4_block6_2_bn     (None, 7, 7, 256)        1,024   conv4_block6_2_c…
(BatchNormalizatio…

conv4_block6_2_relu   (None, 7, 7, 256)            0   conv4_block6_2_b…
(Activation)

max_pooling2d_2       (None, 7, 7,                 0   conv4_block5_out…
(MaxPooling2D)        1024)

conv4_block6_3_conv   (None, 7, 7,           263,168   conv4_block6_2_r…
(Conv2D)              1024)

conv4_block6_out      (None, 7, 7,                 0   max_pooling2d_2[…
(Add)                 1024)                          conv4_block6_3_c…

conv5_block1_preac…   (None, 7, 7,             4,096   conv4_block6_out…
(BatchNormalizatio…   1024)

conv5_block1_preac…   (None, 7, 7,                 0   conv5_block1_pre…
(Activation)          1024)

conv5_block1_1_conv   (None, 7, 7, 512)      524,288   conv5_block1_pre…
(Conv2D)

conv5_block1_1_bn     (None, 7, 7, 512)        2,048   conv5_block1_1_c…
(BatchNormalizatio…

conv5_block1_1_relu   (None, 7, 7, 512)            0   conv5_block1_1_b…
(Activation)

conv5_block1_2_pad    (None, 9, 9, 512)            0   conv5_block1_1_r…
(ZeroPadding2D)

conv5_block1_2_conv   (None, 7, 7, 512)    2,359,296   conv5_block1_2_p…
(Conv2D)

conv5_block1_2_bn     (None, 7, 7, 512)        2,048   conv5_block1_2_c…
(BatchNormalizatio…

conv5_block1_2_relu   (None, 7, 7, 512)            0   conv5_block1_2_b…
(Activation)

conv5_block1_0_conv   (None, 7, 7,         2,099,200   conv5_block1_pre…
(Conv2D)              2048)
```

```
conv5_block1_3_conv   (None, 7, 7,        1,050,624   conv5_block1_2_r…
(Conv2D)              2048)

conv5_block1_out      (None, 7, 7,                0   conv5_block1_0_c…
(Add)                 2048)                           conv5_block1_3_c…

conv5_block2_preac…   (None, 7, 7,            8,192   conv5_block1_out…
(BatchNormalizatio…   2048)

conv5_block2_preac…   (None, 7, 7,                0   conv5_block2_pre…
(Activation)          2048)

conv5_block2_1_conv   (None, 7, 7, 512)   1,048,576   conv5_block2_pre…
(Conv2D)

conv5_block2_1_bn     (None, 7, 7, 512)       2,048   conv5_block2_1_c…
(BatchNormalizatio…

conv5_block2_1_relu   (None, 7, 7, 512)           0   conv5_block2_1_b…
(Activation)

conv5_block2_2_pad    (None, 9, 9, 512)           0   conv5_block2_1_r…
(ZeroPadding2D)

conv5_block2_2_conv   (None, 7, 7, 512)   2,359,296   conv5_block2_2_p…
(Conv2D)

conv5_block2_2_bn     (None, 7, 7, 512)       2,048   conv5_block2_2_c…
(BatchNormalizatio…

conv5_block2_2_relu   (None, 7, 7, 512)           0   conv5_block2_2_b…
(Activation)

conv5_block2_3_conv   (None, 7, 7,        1,050,624   conv5_block2_2_r…
(Conv2D)              2048)

conv5_block2_out      (None, 7, 7,                0   conv5_block1_out…
(Add)                 2048)                           conv5_block2_3_c…

conv5_block3_preac…   (None, 7, 7,            8,192   conv5_block2_out…
(BatchNormalizatio…   2048)

conv5_block3_preac…   (None, 7, 7,                0   conv5_block3_pre…
(Activation)          2048)

conv5_block3_1_conv   (None, 7, 7, 512)   1,048,576   conv5_block3_pre…
(Conv2D)
```

```
conv5_block3_1_bn     (None, 7, 7, 512)        2,048   conv5_block3_1_c…
(BatchNormalizatio…

conv5_block3_1_relu   (None, 7, 7, 512)            0   conv5_block3_1_b…
(Activation)

conv5_block3_2_pad    (None, 9, 9, 512)            0   conv5_block3_1_r…
(ZeroPadding2D)

conv5_block3_2_conv   (None, 7, 7, 512)    2,359,296   conv5_block3_2_p…
(Conv2D)

conv5_block3_2_bn     (None, 7, 7, 512)        2,048   conv5_block3_2_c…
(BatchNormalizatio…

conv5_block3_2_relu   (None, 7, 7, 512)            0   conv5_block3_2_b…
(Activation)

conv5_block3_3_conv   (None, 7, 7,         1,050,624   conv5_block3_2_r…
(Conv2D)              2048)

conv5_block3_out      (None, 7, 7,                 0   conv5_block2_out…
(Add)                 2048)                            conv5_block3_3_c…

post_bn               (None, 7, 7,             8,192   conv5_block3_out…
(BatchNormalizatio…   2048)

post_relu             (None, 7, 7,                 0   post_bn[0][0]
(Activation)          2048)


Total params: 23,564,800 (89.89 MB)


Trainable params: 0 (0.00 B)


Non-trainable params: 23,564,800 (89.89 MB)
```

[ ]:
```
# Agrega capa de clasificación para transformar las características en vectores␣
↪de 2048 elementos

global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(9, 2048)
```

[ ]:
```
# Capa de predicción para transformar las características en una sola predicción por imagen
num_classes = len(class_names)
prediction_layer = tf.keras.layers.Dense(num_classes, activation='softmax') # selecciona softmax debido a que es una clasificación multiclase
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(9, 4)
```

[ ]:
```
# Concatena data augmentation, preprocesamiento, el modelo base, y capas que extraen características.

inputs = tf.keras.Input(shape=(200, 200, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=False) # congela modelo base
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

[ ]:
```
model.summary()
```

```
Model: "functional_1"
```

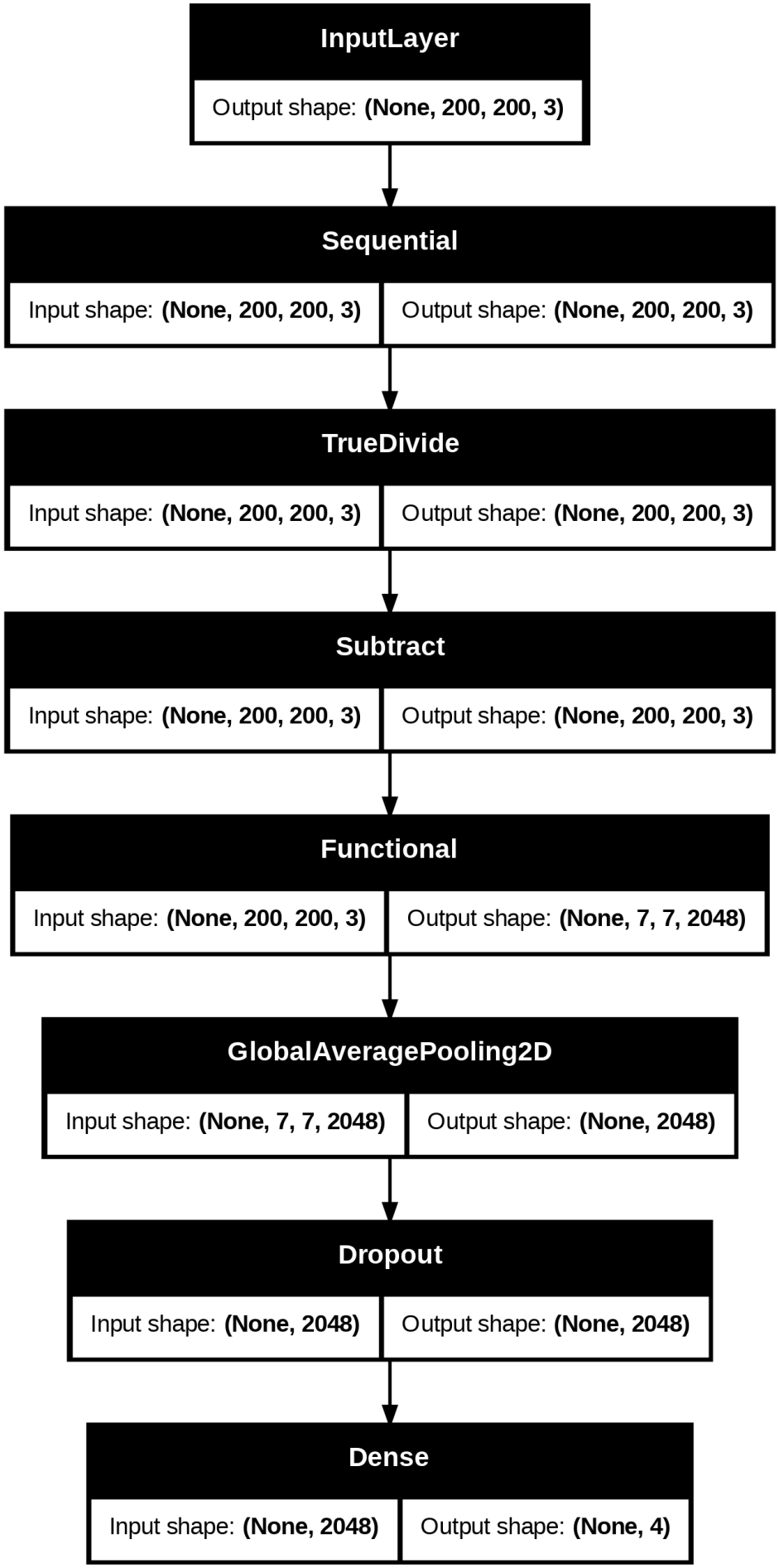
Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 200, 200, 3)	0
sequential (Sequential)	(None, 200, 200, 3)	0
true_divide (TrueDivide)	(None, 200, 200, 3)	0
subtract (Subtract)	(None, 200, 200, 3)	0
resnet50v2 (Functional)	(None, 7, 7, 2048)	23,564,800
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0
dropout (Dropout)	(None, 2048)	0
dense (Dense)	(None, 4)	8,196

```
Total params: 23,572,996 (89.92 MB)

Trainable params: 8,196 (32.02 KB)

Non-trainable params: 23,564,800 (89.89 MB)
```

En este caso se pueden entrenar 8196 parámetros, los cuales corresponden a la capa densa.

[ ]:
```
tf.keras.utils.plot_model(model, show_shapes=True)
```

[ ]:


InputLayer
Output shape: (None, 200, 200, 3)
Sequential
Input shape: (None, 200, 200, 3)
Output shape: (None, 200, 200, 3)
TrueDivide
Input shape: (None, 200, 200, 3)
Output shape: (None, 200, 200, 3)
Subtract
Input shape: (None, 200, 200, 3)
Output shape: (None, 200, 200, 3)
Functional
Input shape: (None, 200, 200, 3)
Output shape: (None, 7, 7, 2048)
GlobalAveragePooling2D
Input shape: (None, 7, 7, 2048)
Output shape: (None, 2048)
Dropout
Input shape: (None, 2048)
Output shape: (None, 2048)
Dense
Input shape: (None, 2048)
Output shape: (None, 4)

En este punto se define el learning rate del modelo y cómo se calculará el parámetro loss.

[ ]:
```
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=False,
              ignore_class=None, reduction='sum_over_batch_size',␣
 ↪name='sparse_categorical_crossentropy'),
              metrics= ['accuracy'])
```

[ ]:
```
initial_epochs = 10

loss0, accuracy0 = model.evaluate(validation_dataset)
```

```
3/3             10s 2s/step -
accuracy: 0.3921 - loss: 1.6392
```

[ ]:
```
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
initial loss: 1.70
initial accuracy: 0.42
```

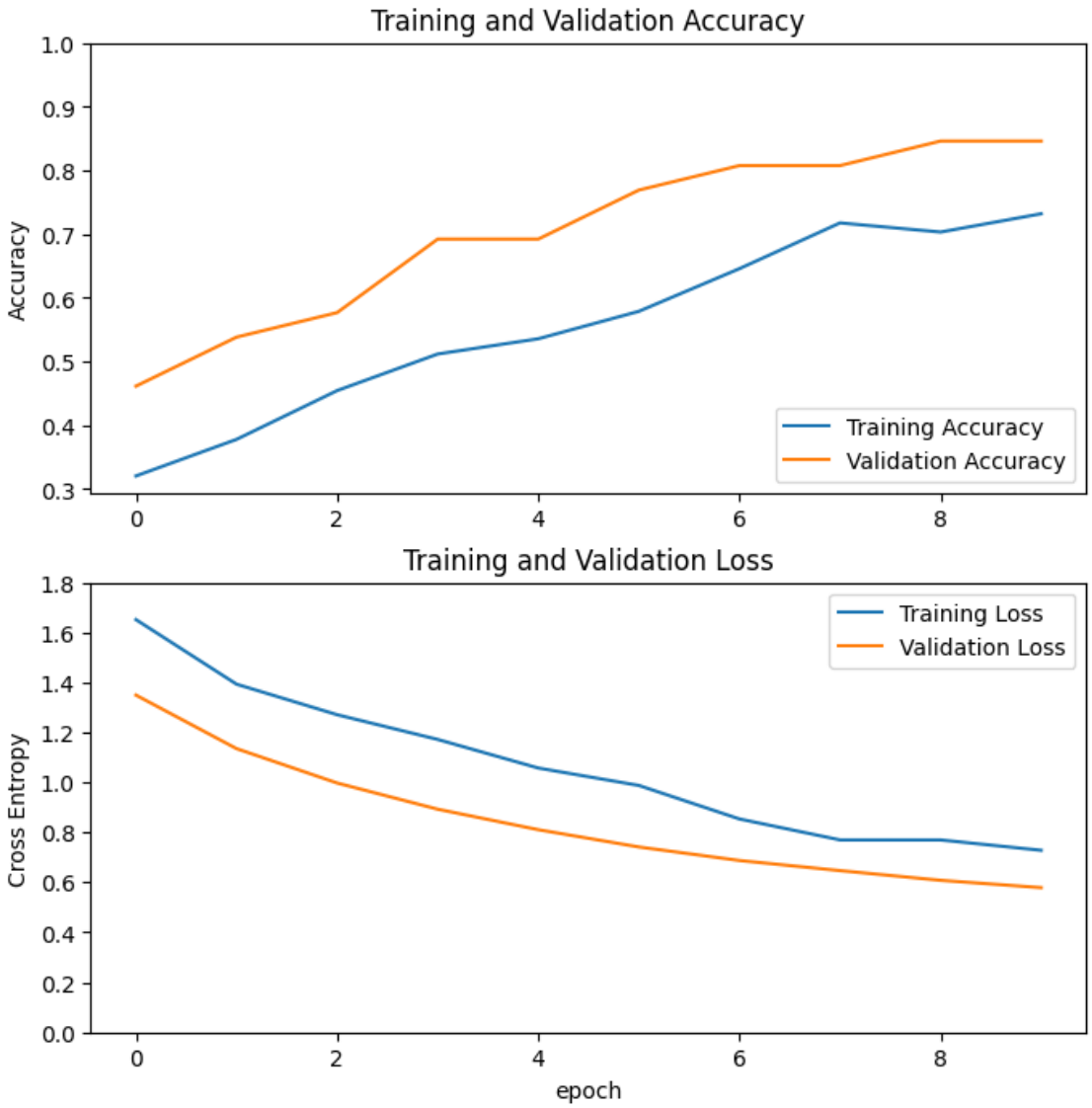
[ ]:
```
# guardar el modelo durante el entrenamiento de la red neuronal
save_checkpoint_1 = tf.keras.callbacks.ModelCheckpoint(
    filepath='./models/model_ResNet50V2.keras', monitor='val_loss',mode='min',␣
 ↪save_best_only=True, verbose=1
    )
```

[ ]:
```
history = model.fit(train_dataset,
                    epochs=initial_epochs,
                    validation_data=validation_dataset)
```

```
Epoch 1/10
24/24             44s 2s/step -
accuracy: 0.2953 - loss: 1.7300 - val_accuracy: 0.4615 - val_loss: 1.3497
Epoch 2/10
24/24             1s 42ms/step -
accuracy: 0.3299 - loss: 1.4832 - val_accuracy: 0.5385 - val_loss: 1.1355
Epoch 3/10
24/24             1s 41ms/step -
accuracy: 0.4017 - loss: 1.3133 - val_accuracy: 0.5769 - val_loss: 0.9979
Epoch 4/10
24/24             1s 43ms/step -
accuracy: 0.4608 - loss: 1.1974 - val_accuracy: 0.6923 - val_loss: 0.8931
```

```
Epoch 5/10
24/24             1s 42ms/step -
accuracy: 0.5343 - loss: 1.0970 - val_accuracy: 0.6923 - val_loss: 0.8111
Epoch 6/10
24/24             1s 42ms/step -
accuracy: 0.6076 - loss: 0.9491 - val_accuracy: 0.7692 - val_loss: 0.7418
Epoch 7/10
24/24             1s 44ms/step -
accuracy: 0.6060 - loss: 0.9115 - val_accuracy: 0.8077 - val_loss: 0.6877
Epoch 8/10
24/24             2s 57ms/step -
accuracy: 0.7069 - loss: 0.8149 - val_accuracy: 0.8077 - val_loss: 0.6472
Epoch 9/10
24/24             2s 43ms/step -
accuracy: 0.6871 - loss: 0.8098 - val_accuracy: 0.8462 - val_loss: 0.6084
Epoch 10/10
24/24             1s 42ms/step -
accuracy: 0.7171 - loss: 0.7291 - val_accuracy: 0.8462 - val_loss: 0.5791
```

[ ]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.8])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Se observa una disminución del parámetro loss tanto para el set de entrenamiento como el de validación.

Fluctúa el parámetro accuracy en el set de entrenamiento entre las epochs 8 y 10.

[ ]:
```
# Permite entrenamiento del modelo base
base_model.trainable = True
```

[ ]:
```
# Verifica cantidad de capas del modelo base
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune  desde esta capa
fine_tune_at = 189
```

```
# Congela capas anteriores a la capa desde la que se realizará el fine-tuning
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable = False
```

```
Number of layers in the base model:  190
```

Debido a que el dataset sólo contiene 209 imágenes el en set de entrenamiento, sólo se descongelaron las últimas capas del modelo para evitar que ocurra overfitting.

También se reduce el learning rate para que el modelo no se sobreajuste al entrenarlo.

[ ]:
```
model.compile(loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=False,
              ignore_class=None, reduction='sum_over_batch_size',␣
 ↪name='sparse_categorical_crossentropy'),
              optimizer = tf.keras.optimizers.
 ↪RMSprop(learning_rate=base_learning_rate/10),
              metrics = ['accuracy'])
```

[ ]:
```
model.summary()
```

```
Model: "functional_1"
```

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 200, 200, 3)	0
sequential (Sequential)	(None, 200, 200, 3)	0
true_divide (TrueDivide)	(None, 200, 200, 3)	0
subtract (Subtract)	(None, 200, 200, 3)	0
resnet50v2 (Functional)	(None, 7, 7, 2048)	23,564,800
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0
dropout (Dropout)	(None, 2048)	0
dense (Dense)	(None, 4)	8,196

```
Total params: 23,572,996 (89.92 MB)
```

Trainable params: 8,196 (32.02 KB)

Non-trainable params: 23,564,800 (89.89 MB)

[ ]:
```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_dataset,
                         epochs=total_epochs,
                         initial_epoch=len(history.epoch),
                         validation_data=validation_dataset,
                         callbacks=[save_checkpoint_1])
```

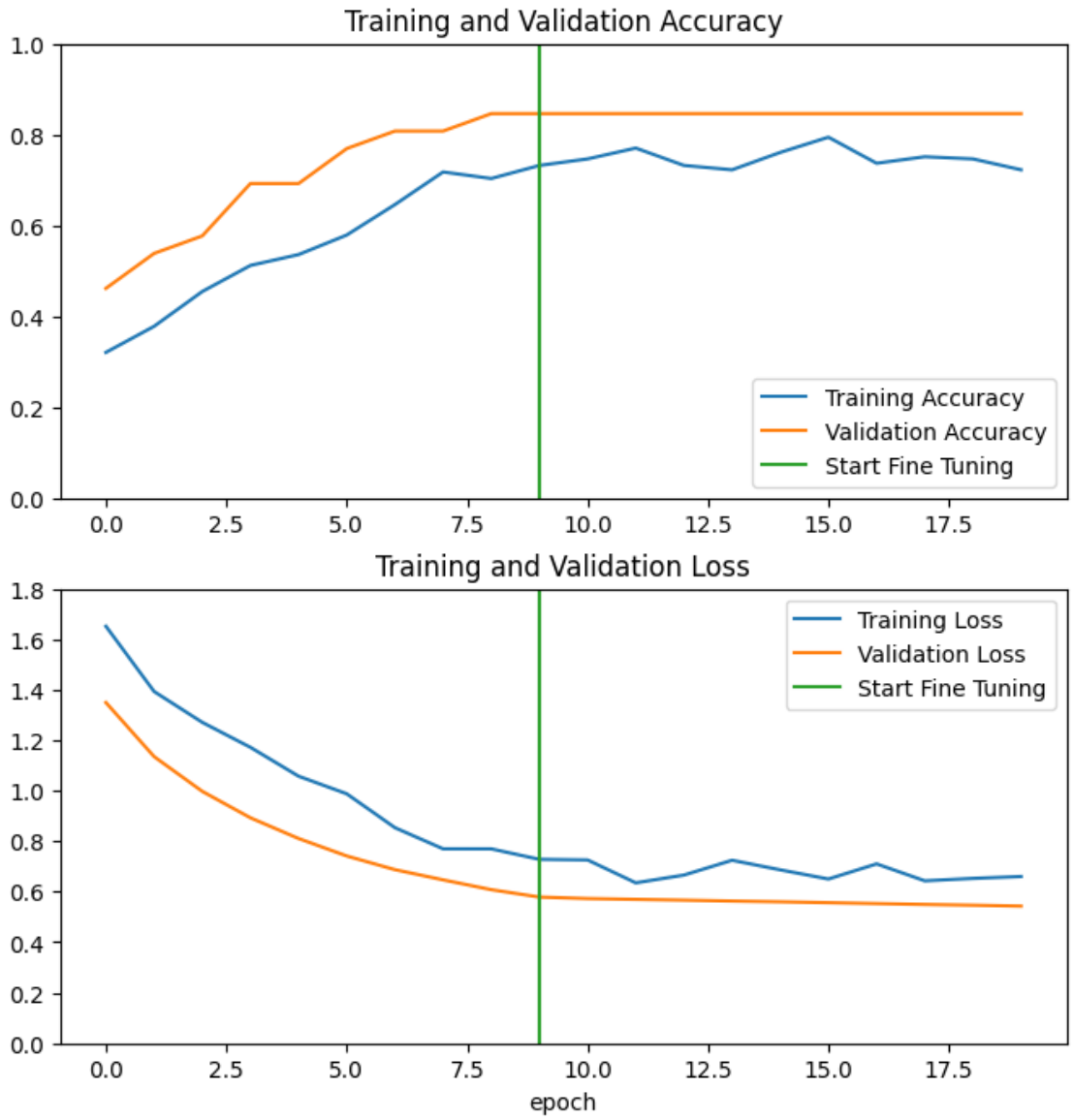
```
Epoch 11/20
24/24             0s 46ms/step -
accuracy: 0.7396 - loss: 0.7188
Epoch 11: val_loss improved from inf to 0.57316, saving model to
./models/model_ResNet50V2.keras
24/24             12s 207ms/step -
accuracy: 0.7399 - loss: 0.7191 - val_accuracy: 0.8462 - val_loss: 0.5732
Epoch 12/20
23/24             0s 38ms/step -
accuracy: 0.7549 - loss: 0.6856
Epoch 12: val_loss improved from 0.57316 to 0.57002, saving model to
./models/model_ResNet50V2.keras
24/24             2s 76ms/step -
accuracy: 0.7562 - loss: 0.6816 - val_accuracy: 0.8462 - val_loss: 0.5700
Epoch 13/20
24/24             0s 51ms/step -
accuracy: 0.7722 - loss: 0.6071
Epoch 13: val_loss improved from 0.57002 to 0.56691, saving model to
./models/model_ResNet50V2.keras
24/24             3s 92ms/step -
accuracy: 0.7706 - loss: 0.6094 - val_accuracy: 0.8462 - val_loss: 0.5669
Epoch 14/20
22/24             0s 41ms/step -
accuracy: 0.7036 - loss: 0.7240
Epoch 14: val_loss improved from 0.56691 to 0.56333, saving model to
./models/model_ResNet50V2.keras
24/24             2s 79ms/step -
accuracy: 0.7062 - loss: 0.7232 - val_accuracy: 0.8462 - val_loss: 0.5633
Epoch 15/20
23/24             0s 55ms/step -
accuracy: 0.7416 - loss: 0.7095
Epoch 15: val_loss improved from 0.56333 to 0.56038, saving model to
./models/model_ResNet50V2.keras
24/24             3s 107ms/step -
```

```
accuracy: 0.7431 - loss: 0.7077 - val_accuracy: 0.8462 - val_loss: 0.5604
Epoch 16/20
23/24             0s 36ms/step -
accuracy: 0.8012 - loss: 0.6141
Epoch 16: val_loss improved from 0.56038 to 0.55685, saving model to
./models/model_ResNet50V2.keras
24/24             2s 75ms/step -
accuracy: 0.8007 - loss: 0.6170 - val_accuracy: 0.8462 - val_loss: 0.5568
Epoch 17/20
23/24             0s 50ms/step -
accuracy: 0.7610 - loss: 0.6963
Epoch 17: val_loss improved from 0.55685 to 0.55329, saving model to
./models/model_ResNet50V2.keras
24/24             2s 89ms/step -
accuracy: 0.7591 - loss: 0.6974 - val_accuracy: 0.8462 - val_loss: 0.5533
Epoch 18/20
23/24             0s 37ms/step -
accuracy: 0.7668 - loss: 0.6316
Epoch 18: val_loss improved from 0.55329 to 0.54977, saving model to
./models/model_ResNet50V2.keras
24/24             2s 74ms/step -
accuracy: 0.7655 - loss: 0.6326 - val_accuracy: 0.8462 - val_loss: 0.5498
Epoch 19/20
24/24             0s 47ms/step -
accuracy: 0.7117 - loss: 0.7073
Epoch 19: val_loss improved from 0.54977 to 0.54681, saving model to
./models/model_ResNet50V2.keras
24/24             3s 88ms/step -
accuracy: 0.7131 - loss: 0.7051 - val_accuracy: 0.8462 - val_loss: 0.5468
Epoch 20/20
23/24             0s 48ms/step -
accuracy: 0.6704 - loss: 0.7589
Epoch 20: val_loss improved from 0.54681 to 0.54319, saving model to
./models/model_ResNet50V2.keras
24/24             3s 97ms/step -
accuracy: 0.6746 - loss: 0.7510 - val_accuracy: 0.8462 - val_loss: 0.5432
```

[ ]:
```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']
```

[ ]:
```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
```

```
plt.ylim([0, 1])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.8])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



El parámetro loss del set de validación se mantiene estable durante el proceso de fine-tuning, pero resulta tener una curva fluctuante en el caso del set de entrenamiento.

El mismo problema se observa en el caso del parámetro accuracy del set de entrenamiento.

[ ]:
```
reconstructed_model = tf.keras.models.load_model('./models/model_ResNet50V2.
↪keras')
```

[ ]:
```
score = reconstructed_model.evaluate(validation_dataset, verbose=False)
print('Val loss:', score[0])
print('Val accuracy:', score[1])
```

```
Val loss: 0.5431869029998779
```

```
Val accuracy: 0.8461538553237915
```

El error del set de validación es menor al del set de prueba y su accuracy es buena, pero debido al pequeño tamaño del dataset utilizado estos resultados pueden fluctuar muchísimo, siendo poco confiables.

[ ]:
```
score = reconstructed_model.evaluate(test_dataset, verbose=False)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.6505529880523682
Test accuracy: 0.7777777910232544
```

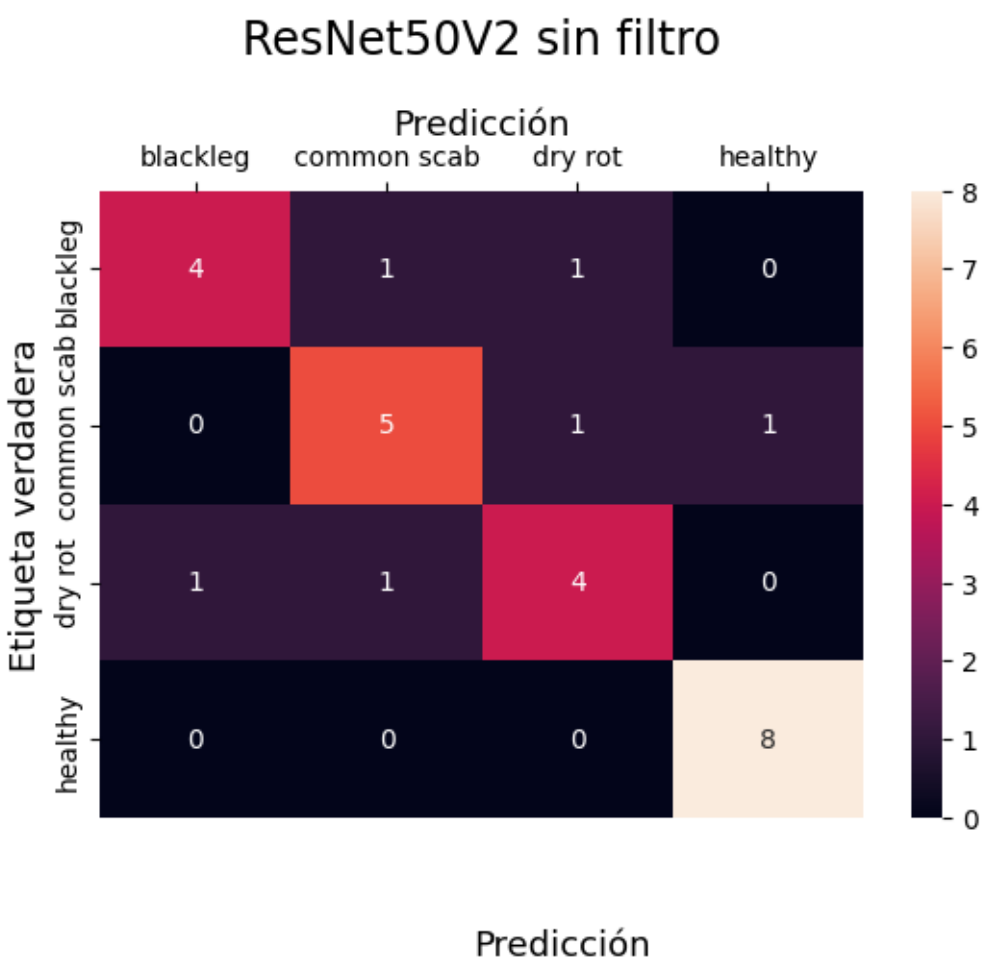
El error del set de prueba es alto, por lo tanto, los resultados de la clasificación de este set no se pueden considerar como confiables a pesar de obtener un valor de accuracy de 0.778. El dataset de prueba sólo contiene 27, cantidad extremadamente baja para un modelo de deep learning.

3.4.1 Matriz de confusión y métricas

[ ]:
```
def get_true_labels_and_predictions(model, dataset):
    true_labels = []
    predictions = []
    for images, labels in dataset:
        true_labels.extend(labels.numpy())
        preds = model.predict(images)
        predictions.extend(np.argmax(preds, axis=1))
    return np.array(true_labels), np.array(predictions)

y_test_1, y_pred_1 = get_true_labels_and_predictions(reconstructed_model,␣
 ↪test_dataset)
cm = confusion_matrix(y_test_1, y_pred_1)
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['blackleg', 'common scab', 'dry rot', 'healthy'],
            yticklabels=['blackleg', 'common scab', 'dry rot', 'healthy'])

plt.ylabel('Etiqueta verdadera', fontsize=13)
plt.title('ResNet50V2 sin filtro', fontsize=17, pad=20)
plt.gca().xaxis.set_label_position('top')
plt.xlabel('Predicción', fontsize=13)
plt.gca().xaxis.tick_top()

plt.gca().figure.subplots_adjust(bottom=0.2)
plt.gca().figure.text(0.5, 0.05, 'Predicción', ha='center', fontsize=13)
plt.show()
```

```
1/1             2s 2s/step
1/1             0s 77ms/step
```

```
1/1                 0s 76ms/step
```



De las seis muestras de papas infectadas con blackleg, cuatro fueron clasificadas correctamente, una fue clasificada como papa infectada por common scab, y otra como papa infectada por dry rot.

Cinco papas fueron clasificadas correctamente como papas infectadas por common scab, una como infectada por dry rot, y otra fue clasificada como una papa sana.

Las ocho papas sanas fueron clasificadas correctamente.

[ ]:
```python
def c_report(y_true, y_pred, class_names):
    report = classification_report(y_true, y_pred, target_names=class_names)
    return report


report_1 = classification_report(y_test_1, y_pred_1, target_names=class_names)

print(f"Classification Report for Model 1:\n{report_1}")
```

```
Classification Report for Model 1:
              precision    recall  f1-score   support

    blackleg       0.80      0.67      0.73         6
 common_scab       0.71      0.71      0.71         7
     dry_rot       0.67      0.67      0.67         6
      potato       0.89      1.00      0.94         8

    accuracy                           0.78        27
   macro avg       0.77      0.76      0.76        27
weighted avg       0.77      0.78      0.77        27
```

Se comprueba que los mejores resultados de precision, recall y F1 score corresponden a la clase potato, lo cual implica que al modelo le es posible distinguir entre las papas sanas y las que están infectadas por alguna bacteria u hongo.

Sin embargo, el modelo no funciona tan bien cuando debe distinguir entre las enfermedades blackleg, common scab y dry rot.

3.4.2 Determinación del tiempo de ejecución del proceso de fine-tuning del modelo ResNet50V2 para el 50% y el 100% de las imágenes del dataset

El tiempo de procesamiento del 50% y 100% de las imágenes del dataset considera el uso de una tarjeta Nvidia T4 para acelerarlo. Sólo se mide cinco veces debido a que la GPU es un recurso costoso de utilizar.

[ ]:
```python
ex_time_nf = []
half_time_nf = []

j = 1

while j < 6:
    start = time.perf_counter()

    fine_tune_epochs = 10
    total_epochs =  initial_epochs + fine_tune_epochs

    history_fine = model.fit(train_dataset,
                         epochs=total_epochs,
                         initial_epoch=len(history.epoch),
                         validation_data=validation_dataset)

    end = time.perf_counter()
    total = end - start  # calcula tiempo total
    print(f"Tiempo transcurrido: {(total)} s")
    ex_time_nf.append(total)
    half_t = (total/2)  # calcula valor para aproximar el tiempo que toma␣
 ↪procesar el 50% de las imágenes
```

```
    half_time_nf.append(half_t)
    j +=1
```

[ ]:
```
print("Tiempo mínimo ", min(ex_time_nf), 's')
print("Tiempo máximo: ", max(ex_time_nf), 's')
print("Tiempo promedio: ", sum(ex_time_nf)/len(ex_time_nf), 's')
print("Desviación estándar del tiempo: ", np.std(ex_time_nf), 's')
```

```
Tiempo mínimo  11.57029012399994 s
Tiempo máximo:  17.26750636099996 s
Tiempo promedio:  14.44040239479998 s
Desviación estándar del tiempo:  2.184918485508314 s
```

[ ]:
```
print("Tiempo mínimo mitad ejecución ", min(half_time_nf), 's')
print("Tiempo máximo mitad ejecución: ", max(half_time_nf), 's')
print("Tiempo promedio mitad ejecución: ", sum(half_time_nf)/len(half_time_nf),␣
 ↪'s')
print("Desviación estándar del tiempo mitad ejecución: ", np.std(half_time_nf),␣
 ↪'s')
```

```
Tiempo mínimo mitad ejecución  5.78514506199997 s
Tiempo máximo mitad ejecución:  8.63375318049998 s
Tiempo promedio mitad ejecución:  7.22020119739999 s
Desviación estándar del tiempo mitad ejecución:  1.092459242754157 s
```

3.5 Fine-tuning modelo ResNet50V2 para dataset filtro Gaussiano

El modelo base utilizado y el proceso de fine-tuning es igual al realizado para el dataset al cual no se le aplicaron filtros.

Solamente se realizó una vez el paso de dividir las carpetas en train, validation y test. En pruebas posteriores se comentó esa parte del código debido a que las carpetas fueron almacenadas en Google Drive.

[ ]:
```
# Dividir carpeta en train, validation y test

path_model = "/content/drive/MyDrive/potato/data_model_g/" # Carpeta para␣
 ↪guardar imágenes
# Crea la carpeta de salida si no existe
if not os.path.exists(path_model):
    os.makedirs(path_model)

splitfolders.ratio("/content/drive/MyDrive/potato/data_g/", output="/content/
 ↪drive/MyDrive/potato/data_model_g/",
    seed=42, ratio=(.8, .1, .1), group_prefix=None, move=False) # default values
```

[ ]:
```
train_dir_g = os.path.join(path_model, 'train')
validation_dir_g = os.path.join(path_model, 'val')
test_dir_g = os.path.join(path_model,'test')
```

```
BATCH_SIZE = 9
IMG_SIZE = (200, 200)

train_dataset_g = tf.keras.utils.image_dataset_from_directory(train_dir_g,
                                                              shuffle=True,
                                                              ␣
 ↪batch_size=BATCH_SIZE,
                                                              image_size=IMG_SIZE)
```

```
Found 209 files belonging to 4 classes.
```

[ ]:
```
validation_dataset_g = tf.keras.utils.
 ↪image_dataset_from_directory(validation_dir_g,
                                                               shuffle=True,
                                                              ␣
 ↪batch_size=BATCH_SIZE,
                                                              ␣
 ↪image_size=IMG_SIZE)
```

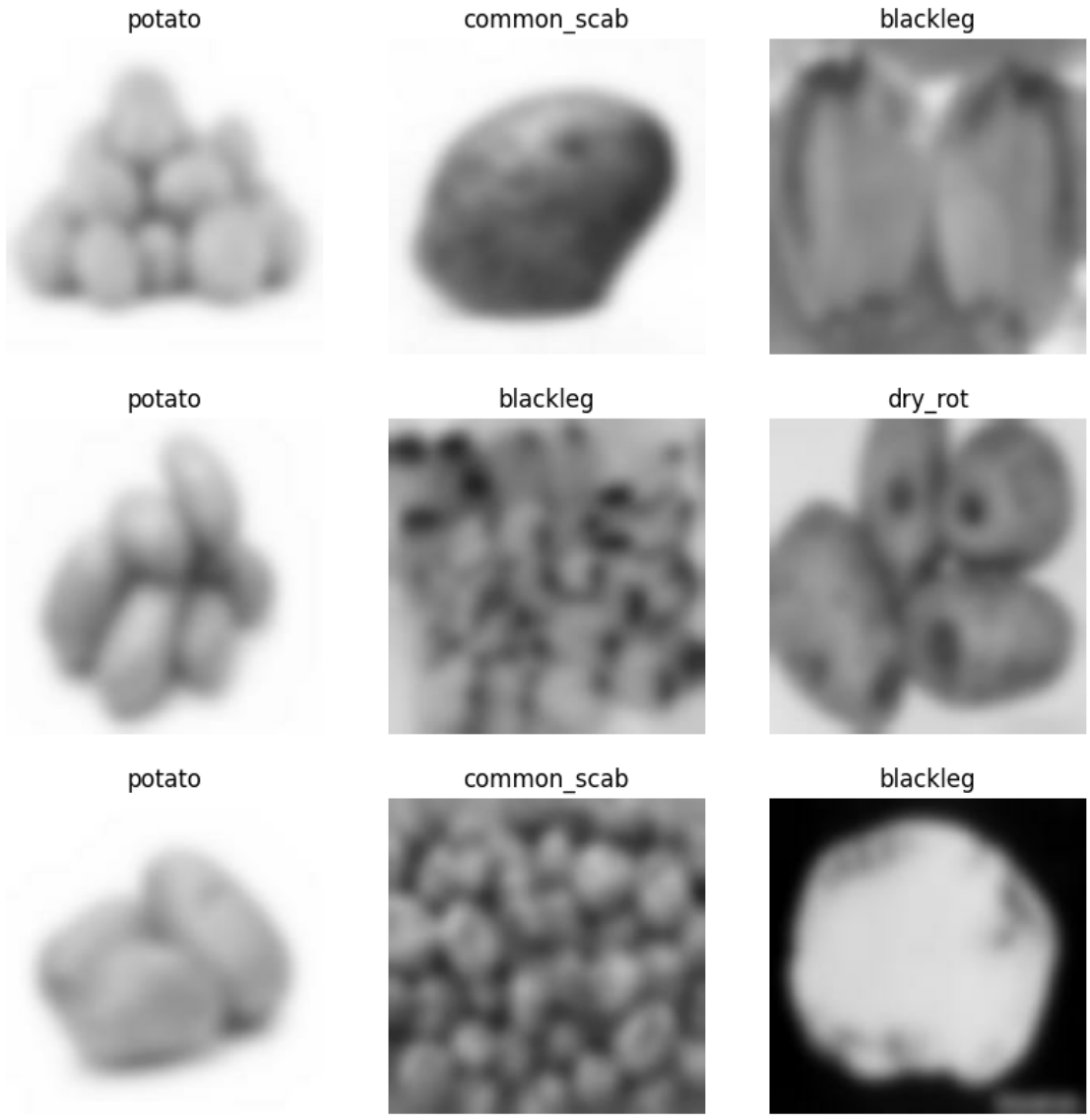
```
Found 26 files belonging to 4 classes.
```

[ ]:
```
test_dataset_g = tf.keras.utils.image_dataset_from_directory(test_dir_g,
                                                               shuffle=True,
                                                              ␣
 ↪batch_size=BATCH_SIZE,
                                                              ␣
 ↪image_size=IMG_SIZE)
```

```
Found 27 files belonging to 4 classes.
```

Se visualiza una muestra de las imágenes del conjunto de entrenamiento y sus respectivas etiquetas en una cuadrícula de 3x3.

[ ]:
```
class_names_g = train_dataset_g.class_names

plt.figure(figsize=(10, 10))
for images, labels in train_dataset_g.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names_g[labels[i]])
    plt.axis("off")
```



[ ]:
```
# Ajusta parametros de memoria automaticamente a la capacidad del computador
AUTOTUNE = tf.data.AUTOTUNE

train_dataset_g = train_dataset_g.prefetch(buffer_size=AUTOTUNE)
validation_dataset_g = validation_dataset_g.prefetch(buffer_size=AUTOTUNE)
test_dataset_g = test_dataset_g.prefetch(buffer_size=AUTOTUNE)
```

[ ]:
```
# Aplica data augmentation al dataset de papas

data_augmentation_g = tf.keras.Sequential([
  tf.keras.layers.RandomFlip('horizontal'),
  tf.keras.layers.RandomRotation(0.1),
```

```
  tf.keras.layers.RandomTranslation(height_factor=0.1, width_factor=0.1),
  tf.keras.layers.RandomContrast(factor=0.1),
])
```

[ ]:
```
# Define el modelo que será aplicado para transformar las imágenes en vectores
preprocess_input = tf.keras.applications.resnet_v2.preprocess_input
```

[ ]:
```
# Crea el modelo base a partir del modelo pre-entrenado a partir del modelo␣
 ↪ResNetV2
IMG_SHAPE = IMG_SIZE + (3,)
base_model_g = tf.keras.applications.ResNet50V2(input_shape=IMG_SHAPE,
                                                include_top=False,
                                                weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/resnet50v2_weights_tf_dim_ordering_tf_kernels_notop.h5
94668760/94668760              3s
0us/step
```

En este paso se convierte a las imágenes en bloques de características de tamaño 7x7x2048.

[ ]:
```
image_batch, label_batch = next(iter(train_dataset_g))
feature_batch = base_model_g(image_batch)
print(feature_batch.shape)
```

```
(9, 7, 7, 2048)
```

[ ]:
```
base_model_g.trainable = False
```

[ ]:
```
# Agrega capa de clasificación para transformar las características en vectores␣
 ↪de 2048 elementos

global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(9, 2048)
```

[ ]:
```
# Capa de predicción para transformar las características en una sola␣
 ↪predicción por imagen
num_classes = len(class_names_g)
prediction_layer = tf.keras.layers.Dense(num_classes, activation='softmax')
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(9, 4)
```

[ ]:
```
# Concatena data augmentation, preprocesamiento, el modelo base, y capas que␣
 ↪extraen características.
```

```
inputs = tf.keras.Input(shape=(200, 200, 3))
x = data_augmentation_g(inputs)
x = preprocess_input(x)
x = base_model_g(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model_g = tf.keras.Model(inputs, outputs)
```

[ ]:
```
base_learning_rate = 0.0001
model_g.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=False,
              ignore_class=None, reduction='sum_over_batch_size',␣
 ↪name='sparse_categorical_crossentropy'),
              metrics = ['accuracy'])
```

[ ]:
```
initial_epochs = 10

loss0, accuracy0 = model_g.evaluate(validation_dataset_g)
```

```
3/3             11s 2s/step -
accuracy: 0.3697 - loss: 1.5502
```

[ ]:
```
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
initial loss: 1.54
initial accuracy: 0.46
```

[ ]:
```
# guardar el modelo durante el entrenamiento de la red neuronal.
save_checkpoint_2 = tf.keras.callbacks.ModelCheckpoint(
    filepath='./models/model_g_ResNet50V2.keras',␣
 ↪monitor='val_loss',mode='min', save_best_only=True, verbose=1
    )
```

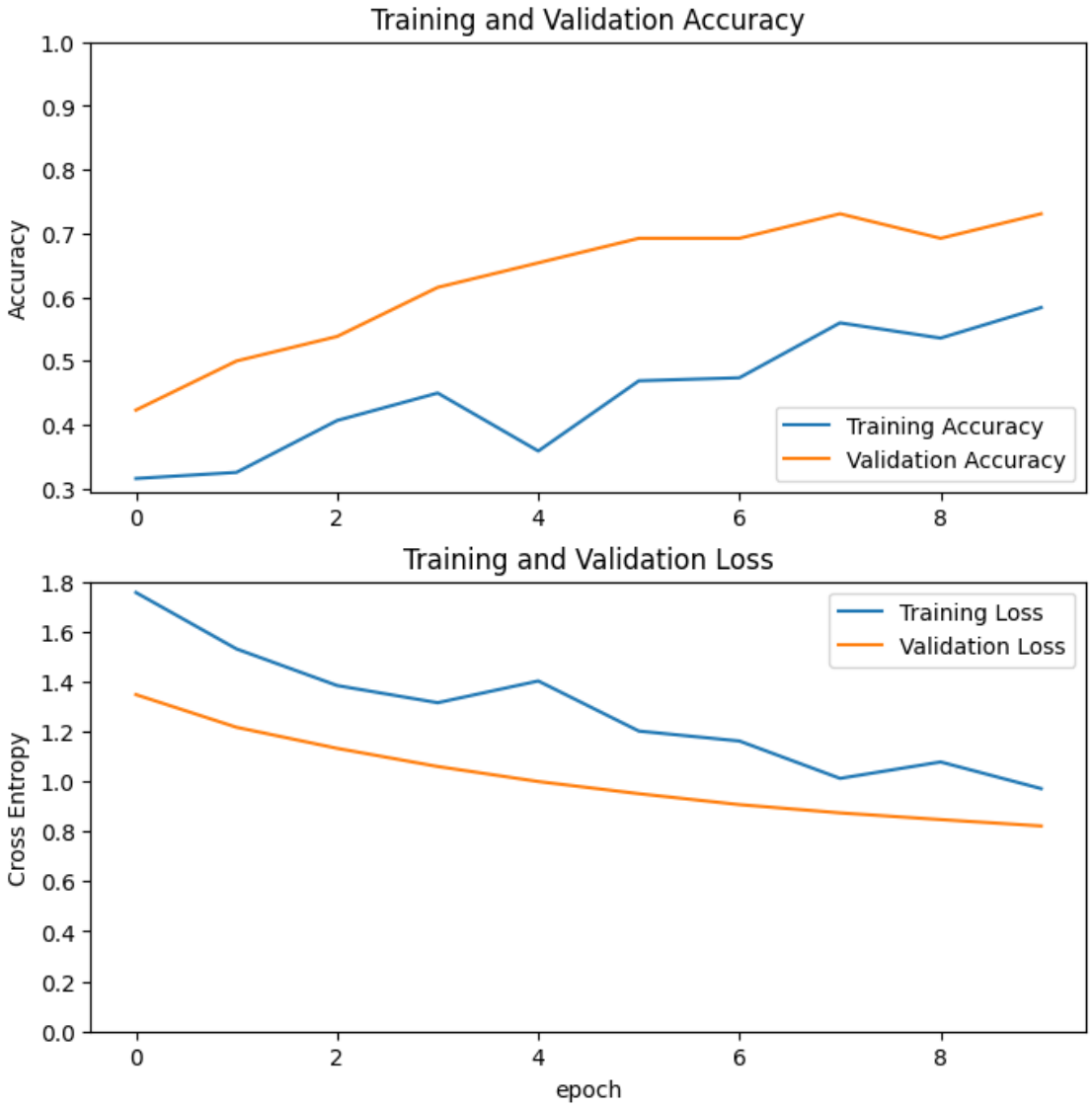
[ ]:
```
history = model_g.fit(train_dataset_g,
                      epochs=initial_epochs,
                      validation_data=validation_dataset_g)
```

```
Epoch 1/10
24/24             56s 2s/step -
accuracy: 0.3205 - loss: 1.8198 - val_accuracy: 0.4231 - val_loss: 1.3477
Epoch 2/10
24/24             33s 45ms/step -
accuracy: 0.3235 - loss: 1.6029 - val_accuracy: 0.5000 - val_loss: 1.2170
Epoch 3/10
```

```
24/24             1s 43ms/step -
accuracy: 0.4113 - loss: 1.3745 - val_accuracy: 0.5385 - val_loss: 1.1327
Epoch 4/10
24/24             1s 43ms/step -
accuracy: 0.4144 - loss: 1.4035 - val_accuracy: 0.6154 - val_loss: 1.0600
Epoch 5/10
24/24             1s 42ms/step -
accuracy: 0.3426 - loss: 1.4553 - val_accuracy: 0.6538 - val_loss: 0.9998
Epoch 6/10
24/24             1s 44ms/step -
accuracy: 0.4804 - loss: 1.1978 - val_accuracy: 0.6923 - val_loss: 0.9512
Epoch 7/10
24/24             1s 48ms/step -
accuracy: 0.4587 - loss: 1.2008 - val_accuracy: 0.6923 - val_loss: 0.9074
Epoch 8/10
24/24             1s 50ms/step -
accuracy: 0.5320 - loss: 1.0640 - val_accuracy: 0.7308 - val_loss: 0.8744
Epoch 9/10
24/24             1s 43ms/step -
accuracy: 0.4595 - loss: 1.1820 - val_accuracy: 0.6923 - val_loss: 0.8475
Epoch 10/10
24/24             1s 42ms/step -
accuracy: 0.5835 - loss: 0.9752 - val_accuracy: 0.7308 - val_loss: 0.8220
```

[ ]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.8])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
```

```
plt.show()
```



La curva de pérdida (loss) del conjunto de validación disminuye sin mostrar fluctuaciones, mientras que la del conjunto de entrenamiento presenta cambios en su pendiente y fluctuaciones entre las epoch 6 y 10. Además, estas fluctuaciones ocurren en la curva de la métrica accuracy del mismo conjunto.

[ ]:
```
# Permite entrenamiento del modelo base
base_model_g.trainable = True
```

[ ]:
```
# Verifica cantidad de capas del modelo base
print("Number of layers in the base model: ", len(base_model_g.layers))
```

```
# Fine-tune  desde esta capa
fine_tune_at = 189

# Congela capas anteriores a la capa desde la que se realizará el fine-tuning
for layer in base_model_g.layers[:fine_tune_at]:
  layer.trainable = False
```

```
Number of layers in the base model:  190
```

[ ]:
```
model_g.compile(loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=False,
             ignore_class=None, reduction='sum_over_batch_size',␣
 ↪name='sparse_categorical_crossentropy'),
             optimizer = tf.keras.optimizers.
 ↪RMSprop(learning_rate=base_learning_rate/10),
             metrics = ['accuracy'])
```

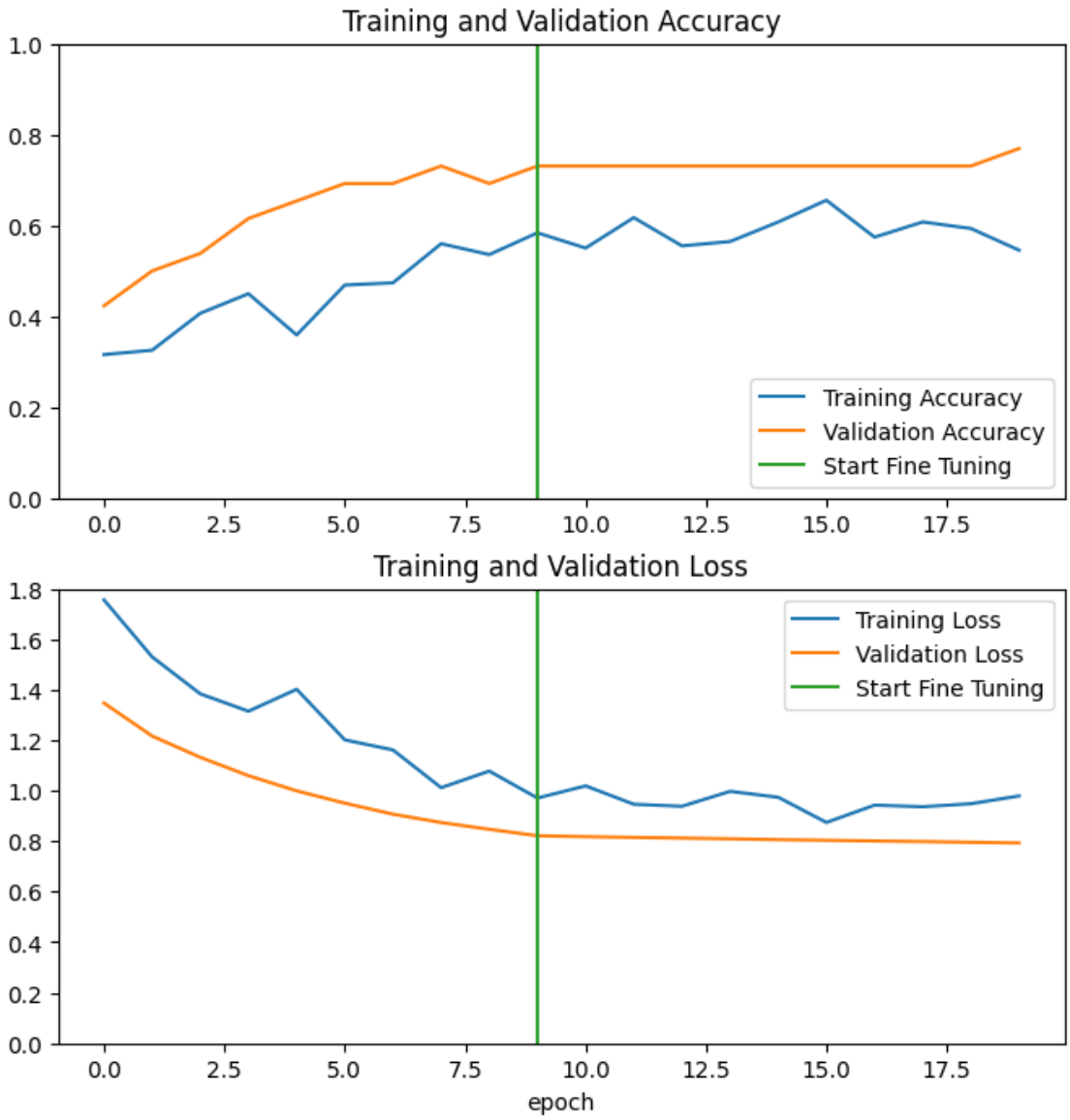
[ ]:
```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model_g.fit(train_dataset_g,
                         epochs=total_epochs,
                         initial_epoch=len(history.epoch),
                         validation_data=validation_dataset_g,
                         callbacks=[save_checkpoint_2])
```

```
Epoch 11/20
23/24             0s 40ms/step -
accuracy: 0.5104 - loss: 1.0647
Epoch 11: val_loss improved from inf to 0.81845, saving model to
./models/model_g_ResNet50V2.keras
24/24             11s 176ms/step -
accuracy: 0.5136 - loss: 1.0611 - val_accuracy: 0.7308 - val_loss: 0.8184
Epoch 12/20
23/24             0s 42ms/step -
accuracy: 0.6261 - loss: 0.9634
Epoch 12: val_loss improved from 0.81845 to 0.81545, saving model to
./models/model_g_ResNet50V2.keras
24/24             2s 90ms/step -
accuracy: 0.6254 - loss: 0.9621 - val_accuracy: 0.7308 - val_loss: 0.8154
Epoch 13/20
23/24             0s 37ms/step -
accuracy: 0.5210 - loss: 0.9711
Epoch 13: val_loss improved from 0.81545 to 0.81284, saving model to
./models/model_g_ResNet50V2.keras
24/24             2s 83ms/step -
accuracy: 0.5237 - loss: 0.9685 - val_accuracy: 0.7308 - val_loss: 0.8128
Epoch 14/20
```

```
22/24             0s 48ms/step -
accuracy: 0.5140 - loss: 1.0415
Epoch 14: val_loss improved from 0.81284 to 0.80986, saving model to
./models/model_g_ResNet50V2.keras
24/24             2s 84ms/step -
accuracy: 0.5201 - loss: 1.0364 - val_accuracy: 0.7308 - val_loss: 0.8099
Epoch 15/20
24/24             0s 37ms/step -
accuracy: 0.5834 - loss: 1.0402
Epoch 15: val_loss improved from 0.80986 to 0.80645, saving model to
./models/model_g_ResNet50V2.keras
24/24             2s 75ms/step -
accuracy: 0.5844 - loss: 1.0376 - val_accuracy: 0.7308 - val_loss: 0.8065
Epoch 16/20
24/24             0s 52ms/step -
accuracy: 0.6114 - loss: 0.9026
Epoch 16: val_loss improved from 0.80645 to 0.80407, saving model to
./models/model_g_ResNet50V2.keras
24/24             3s 90ms/step -
accuracy: 0.6131 - loss: 0.9015 - val_accuracy: 0.7308 - val_loss: 0.8041
Epoch 17/20
23/24             0s 50ms/step -
accuracy: 0.5271 - loss: 1.0269
Epoch 17: val_loss improved from 0.80407 to 0.80136, saving model to
./models/model_g_ResNet50V2.keras
24/24             2s 100ms/step -
accuracy: 0.5309 - loss: 1.0202 - val_accuracy: 0.7308 - val_loss: 0.8014
Epoch 18/20
23/24             0s 57ms/step -
accuracy: 0.6061 - loss: 0.9771
Epoch 18: val_loss improved from 0.80136 to 0.79911, saving model to
./models/model_g_ResNet50V2.keras
24/24             2s 93ms/step -
accuracy: 0.6062 - loss: 0.9739 - val_accuracy: 0.7308 - val_loss: 0.7991
Epoch 19/20
23/24             0s 50ms/step -
accuracy: 0.5378 - loss: 1.0705
Epoch 19: val_loss improved from 0.79911 to 0.79615, saving model to
./models/model_g_ResNet50V2.keras
24/24             2s 87ms/step -
accuracy: 0.5422 - loss: 1.0608 - val_accuracy: 0.7308 - val_loss: 0.7961
Epoch 20/20
24/24             0s 46ms/step -
accuracy: 0.5193 - loss: 1.0086
Epoch 20: val_loss improved from 0.79615 to 0.79352, saving model to
./models/model_g_ResNet50V2.keras
24/24             2s 83ms/step -
accuracy: 0.5203 - loss: 1.0074 - val_accuracy: 0.7692 - val_loss: 0.7935
```

[ ]:
```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']
```

[ ]:
```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0, 1])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.8])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Luego de iniciar el proceso de fine-tuning la curva de pérdida (loss) del conjunto de validación se mantiene estable.

La curva de pérdida del conjunto de entrenamiento es ruidosa.

[ ]:
```
reconstructed_model_2 = tf.keras.models.load_model('./models/model_g_ResNet50V2.
↪keras')
```

[ ]:
```
score2 = reconstructed_model_2.evaluate(validation_dataset_g, verbose=False)
print('Val loss:', score2[0])
print('Val accuracy:', score2[1])
```

```
Val loss: 0.7935184836387634
```

```
Val accuracy: 0.7692307829856873
```

```
[ ]: score2 = reconstructed_model_2.evaluate(test_dataset_g, verbose=False)
print('Test loss:', score2[0])
print('Test accuracy:', score2[1])
```

```
Test loss: 1.0617337226867676
Test accuracy: 0.5185185074806213
```

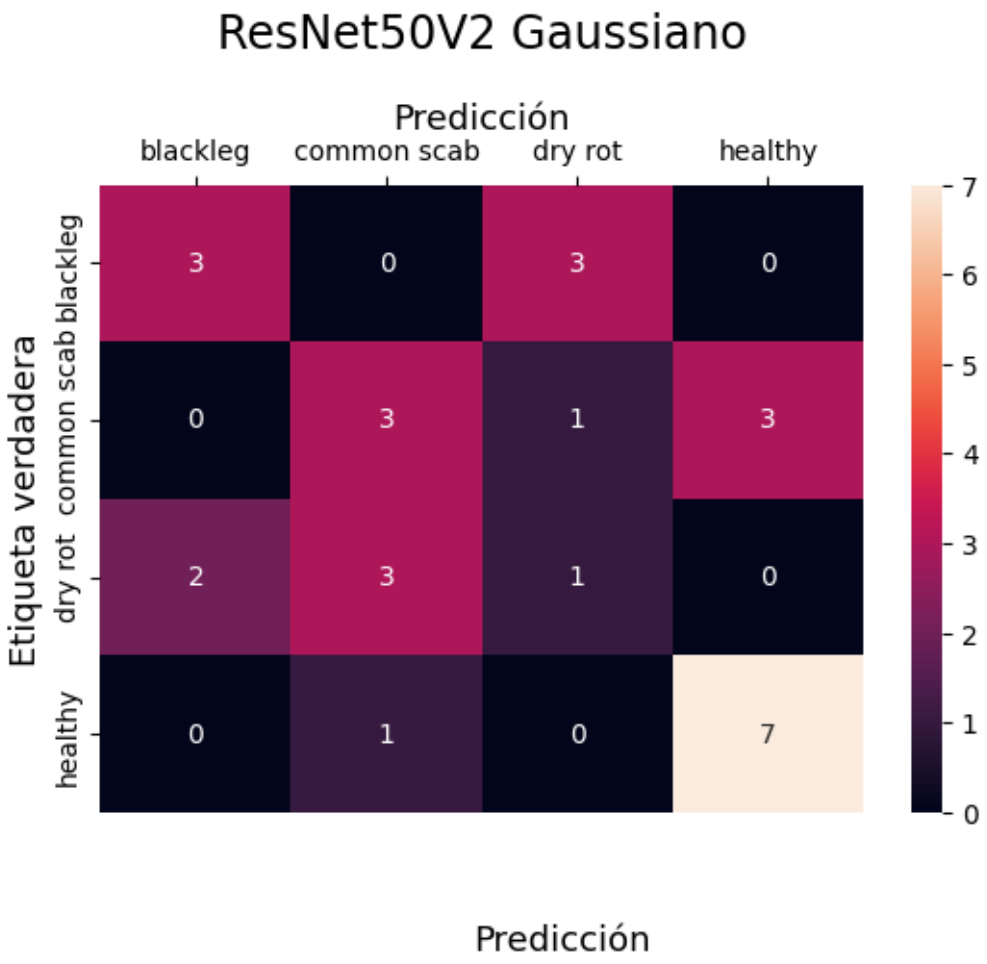
El valor de la pérdida del conjunto de prueba es muy alta. Los resultados son obtenidos completamente al azar en este caso, debido a que el modelo no pudo aprender correctamente.

Es posible suponer que los problemas principales son el tamaño del dataset, el cual es demasiado pequeño, y que las fotografías a las cuales se les aplicó desenfoque Gaussiano no son similares a las imágenes del dataset ImageNet, debido a que se trata de imágenes en escala de grises.

3.5.1 Matriz de confusión y métricas

```
[ ]: def get_true_labels_and_predictions(model, dataset):
    true_labels = []
    predictions = []
    for images, labels in dataset:
        true_labels.extend(labels.numpy())
        preds = model.predict(images)
        predictions.extend(np.argmax(preds, axis=1))
    return np.array(true_labels), np.array(predictions)

y_test_2, y_pred_2 = get_true_labels_and_predictions(reconstructed_model_2,␣
 ↪test_dataset_g)
cm = confusion_matrix(y_test_2, y_pred_2)
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['blackleg', 'common scab', 'dry rot', 'healthy'],
            yticklabels=['blackleg', 'common scab', 'dry rot', 'healthy'])

plt.ylabel('Etiqueta verdadera', fontsize=13)
plt.title('ResNet50V2 Gaussiano', fontsize=17, pad=20)
plt.gca().xaxis.set_label_position('top')
plt.xlabel('Predicción', fontsize=13)
plt.gca().xaxis.tick_top()

plt.gca().figure.subplots_adjust(bottom=0.2)
plt.gca().figure.text(0.5, 0.05, 'Predicción', ha='center', fontsize=13)
plt.show()
```

```
1/1             2s 2s/step
1/1             0s 80ms/step
1/1             0s 88ms/step
```



Tres de las papas con blackleg fueron clasificadas como papas infectadas por dry rot.

Tres papas fueron clasificadas correctamente como papas con common scab, determinando erróneamente que una papa fue infectada por dry rot, y tres papas fueron etiquetadas como papas sanas.

Dos de las papas infectadas por dry rot fueron etiquetadas como papas infectadas por blackleg, y una fue clasificada como papa sana.

Casi todas las papas sanas fueron etiquetadas correctamente, excepto por una que fue clasificada como infectada por common scab.

[ ]:
```
def c_report(y_true, y_pred, class_names):
    report = classification_report(y_true, y_pred, target_names=class_names)
    return report

report_2 = c_report(y_test_2, y_pred_2, class_names_g)

print(f"Classification Report for Model 2:\n{report_2}")
```

```
Classification Report for Model 2:
              precision    recall  f1-score   support

    blackleg       0.60      0.50      0.55         6
 common_scab       0.43      0.43      0.43         7
     dry_rot       0.20      0.17      0.18         6
      potato       0.70      0.88      0.78         8

    accuracy                           0.52        27
   macro avg       0.48      0.49      0.48        27
weighted avg       0.50      0.52      0.50        27
```

En este modelo se observa que en general no distingue tan bien las clases del dataset de papas. Los peores resultados de precision, recall y accuracy corresponden a la clase dry rot.

Se puede asumir que las imágenes difuminadas en escala de grises no permiten diferenciar correctamente las enfermedades presentes en las papas, siendo este problema muy notorio al analizar los resultados de las papas afectadas por dry rot.

3.5.2 Determinación del tiempo de ejecución del proceso de fine-tuning del modelo ResNet50V2 para el 50% y el 100% de las imágenes del dataset

El tiempo de procesamiento del 50% y 100% de las imágenes del dataset considera el uso de una tarjeta Nvidia T4 para acelerarlo. Sólo se mide cinco veces debido a que la GPU es un recurso costoso de utilizar.

[ ]:
```python
ex_time = []
half_time = []

j = 1

while j < 6:
    start = time.perf_counter()


    fine_tune_epochs = 10
    total_epochs =  initial_epochs + fine_tune_epochs

    history_fine = model_g.fit(train_dataset_g,
                         epochs=total_epochs,
                         initial_epoch=len(history.epoch),
                         validation_data=validation_dataset_g)

    end = time.perf_counter()
    total = end - start
    print(f"Tiempo transcurrido: {(total)} s")
    ex_time.append(total)
    half_t = (total/2)
```

```
    half_time.append(half_t)
    j +=1
```

[ ]:
```
print("Tiempo mínimo ", min(ex_time), 's')
print("Tiempo máximo: ", max(ex_time), 's')
print("Tiempo promedio: ", sum(ex_time)/len(ex_time),'s')
print("Desviación estándar del tiempo: ", np.std(ex_time), 's')
```

```
Tiempo mínimo  11.599354496999922 s
Tiempo máximo:  13.490979503000062 s
Tiempo promedio:  12.412873184599993 s
Desviación estándar del tiempo:  0.7685996801703832 s
```

[ ]:
```
print("Tiempo mínimo mitad ejecución ", min(half_time), 's')
print("Tiempo máximo mitad ejecución: ", max(half_time), 's')
print("Tiempo promedio mitad ejecución: ", sum(half_time)/len(half_time), 's')
print("Desviación estándar del tiempo mitad ejecución: ", np.std(half_time),␣
 ↪'s')
```

```
Tiempo mínimo mitad ejecución  5.799677248499961 s
Tiempo máximo mitad ejecución:  6.745489751500031 s
Tiempo promedio mitad ejecución:  6.206436592299997 s
Desviación estándar del tiempo mitad ejecución:  0.3842998400851916 s
```

3.6 Fine-tuning modelo ResNet50V2 para dataset filtro Sobel

El procedimiento realizado en esta sección es el mismo utilizado para los datasets sin filtrar y al cual se le aplicó desenfoque Gaussiano.

Solamente se realizó una vez el paso de dividir las carpetas en train, validation y test. En pruebas posteriores se comentó esa parte del código debido a que las carpetas fueron almacenadas en Google Drive.

[ ]:
```
# Dividir carpeta en train, validation y test

path_model = "/content/drive/MyDrive/potato/data_model_s/" # Carpeta para␣
 ↪guardar imágenes
# Crea la carpeta de salida si no existe
if not os.path.exists(path_model):
    os.makedirs(path_model)

splitfolders.ratio("/content/drive/MyDrive/potato/data_s/", output="/content/
 ↪drive/MyDrive/potato/data_model_s/",
    seed=42, ratio=(.8, .1, .1), group_prefix=None, move=False) # default values
```

[ ]:
```
train_dir_s = os.path.join(path_model, 'train')
validation_dir_s = os.path.join(path_model, 'val')
test_dir_s = os.path.join(path_model,'test')
```

```
BATCH_SIZE = 9
IMG_SIZE = (200, 200)

train_dataset_s = tf.keras.utils.image_dataset_from_directory(train_dir_s,
                                                              shuffle=True,
                                                              ␣
 ↪batch_size=BATCH_SIZE,
                                                              image_size=IMG_SIZE)
```

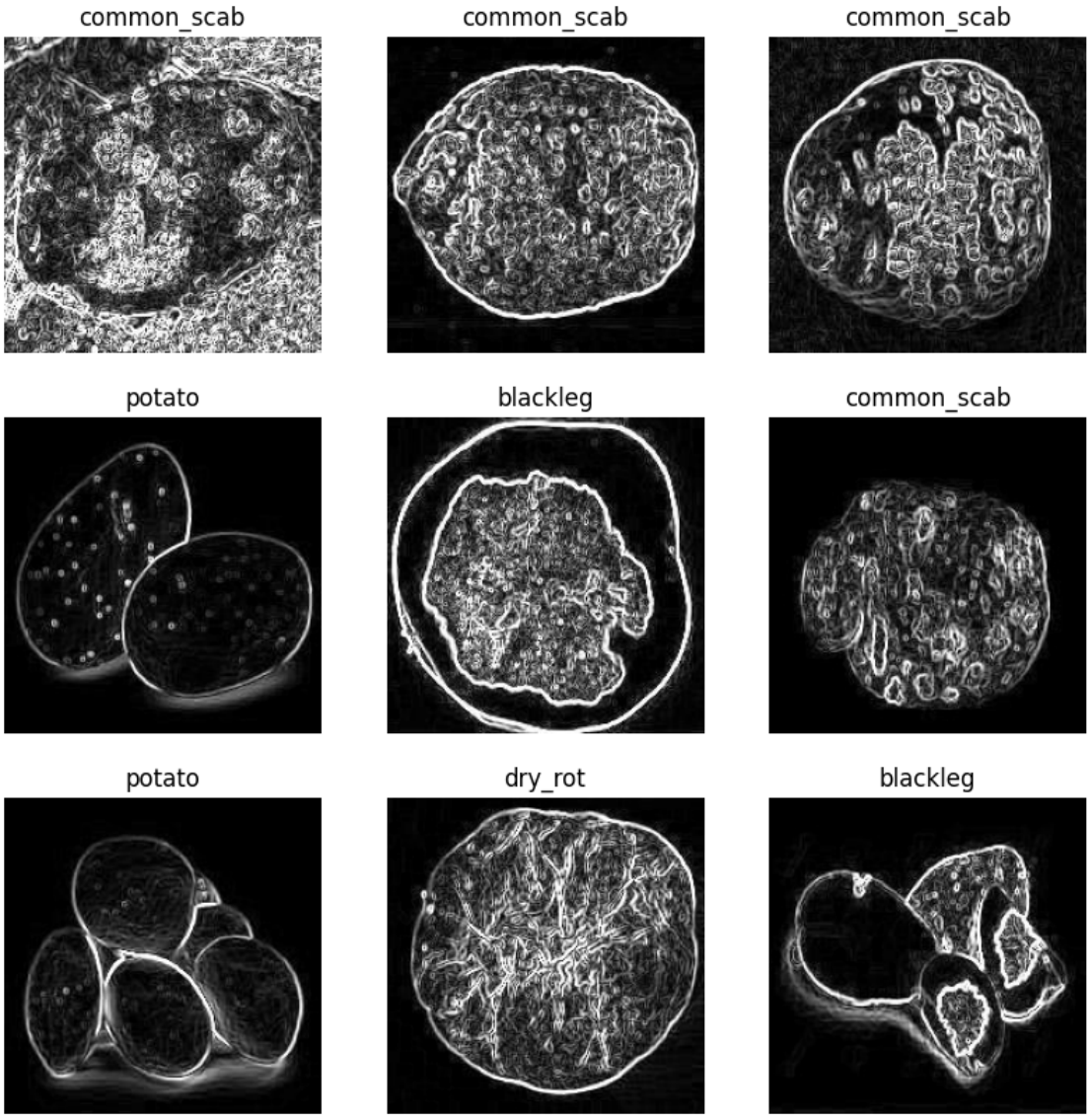
```
Found 209 files belonging to 4 classes.
```

[ ]:
```
validation_dataset_s = tf.keras.utils.
 ↪image_dataset_from_directory(validation_dir_s,
                                                               shuffle=True,
                                                              ␣
 ↪batch_size=BATCH_SIZE,
                                                              ␣
 ↪image_size=IMG_SIZE)

test_dataset_s = tf.keras.utils.image_dataset_from_directory(test_dir_s,
                                                               shuffle=True,
                                                              ␣
 ↪batch_size=BATCH_SIZE,
                                                              ␣
 ↪image_size=IMG_SIZE)
```

```
Found 26 files belonging to 4 classes.
Found 27 files belonging to 4 classes.
```

Se visualiza una muestra de las imágenes del conjunto de entrenamiento y sus respectivas etiquetas en una cuadrícula de 3x3.

[ ]:
```
class_names_s = train_dataset_s.class_names

plt.figure(figsize=(10, 10))
for images, labels in train_dataset_s.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names_s[labels[i]])
    plt.axis("off")
```



En el caso del primer ejemplo de la papa infectada por common scab es prácticamente imposible distinguir a la papa del fondo de la imagen. Es muy probable que el modelo ResNet50V2 no sea capaz de clasificar esta imagen correctamente debido a dicho problema.

[ ]:
```
# Ajusta parametros de memoria automaticamente a la capacidad del computador
AUTOTUNE = tf.data.AUTOTUNE

train_dataset_s = train_dataset_s.prefetch(buffer_size=AUTOTUNE)
validation_dataset_s = validation_dataset_s.prefetch(buffer_size=AUTOTUNE)
test_dataset_s = test_dataset_s.prefetch(buffer_size=AUTOTUNE)
```

[ ]:
```
# Aplica data augmentation al dataset de papas

data_augmentation_s = tf.keras.Sequential([
  tf.keras.layers.RandomFlip('horizontal'),
  tf.keras.layers.RandomRotation(0.1),
  tf.keras.layers.RandomTranslation(height_factor=0.1, width_factor=0.1),
  tf.keras.layers.RandomContrast(factor=0.1),
])
```

[ ]:
```
# Define el modelo que será aplicado para transformar las imágenes en vectores
preprocess_input = tf.keras.applications.resnet_v2.preprocess_input
```

[ ]:
```
# Crea el modelo base a partir del modelo pre-entrenado a partir del modelo␣
 ↪ResNet50V2
IMG_SHAPE = IMG_SIZE + (3,)
base_model_s = tf.keras.applications.ResNet50V2(input_shape=IMG_SHAPE,
                                                include_top=False,
                                                weights='imagenet')
```

[ ]:
```
image_batch, label_batch = next(iter(train_dataset_s))
feature_batch = base_model_s(image_batch)
print(feature_batch.shape)
```

```
(9, 7, 7, 2048)
```

[ ]:
```
base_model_s.trainable = False
```

[ ]:
```
# Agrega capa de clasificación para transformar las características en vectores␣
 ↪de 2048 elementos

global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(9, 2048)
```

[ ]:
```
# Capa de predicción para transformar las características en una sola␣
 ↪predicción por imagen
num_classes = len(class_names_s)
prediction_layer = tf.keras.layers.Dense(num_classes, activation='softmax')
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(9, 4)
```

[ ]:
```
# Concatena data augmentation, preprocesamiento, el modelo base, y capas que␣
 ↪extraen características.

inputs = tf.keras.Input(shape=(200, 200, 3))
```

```
x = data_augmentation_s(inputs)
x = preprocess_input(x)
x = base_model_s(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model_s = tf.keras.Model(inputs, outputs)
```

[ ]:
```
base_learning_rate = 0.0001
model_s.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=False,
              ignore_class=None, reduction='sum_over_batch_size',␣
 ↪name='sparse_categorical_crossentropy'),
              metrics = ['accuracy'])
```

[ ]:
```
initial_epochs = 10

loss0, accuracy0 = model_s.evaluate(validation_dataset_s)
```

```
3/3             10s 2s/step -
accuracy: 0.3013 - loss: 1.8648
```

[ ]:
```
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
initial loss: 2.01
initial accuracy: 0.27
```

[ ]:
```
# guardar el modelo durante el entrenamiento de la red neuronal.
save_checkpoint_3 = tf.keras.callbacks.ModelCheckpoint(
    filepath='./models/model_s_ResNet50V2.keras',␣
 ↪monitor='val_loss',mode='min', save_best_only=True, verbose=1
    )
```

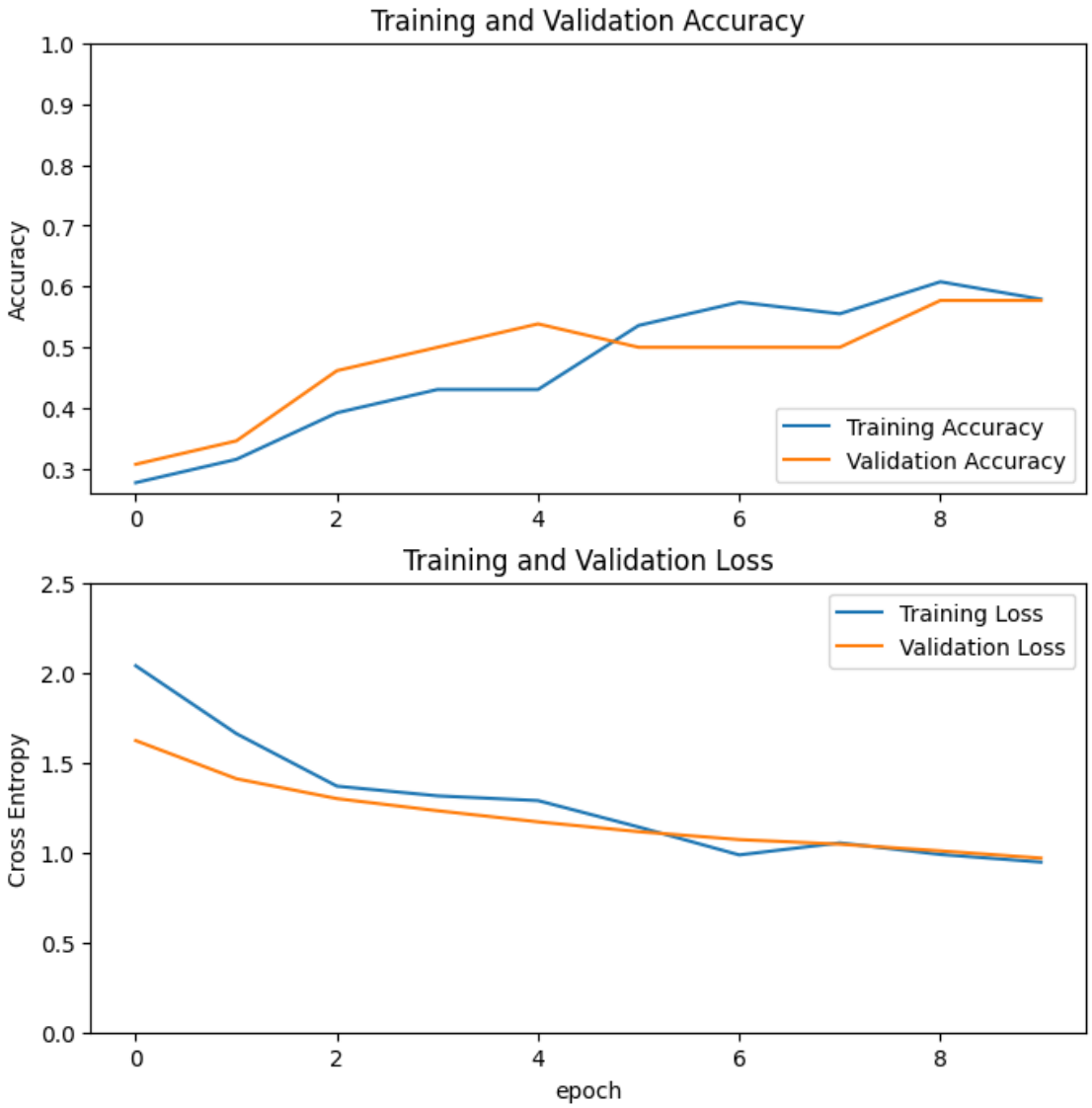
[ ]:
```
history = model_s.fit(train_dataset_s,
                      epochs=initial_epochs,
                      validation_data=validation_dataset_s)
```

```
Epoch 1/10
24/24             55s 2s/step -
accuracy: 0.2583 - loss: 2.2143 - val_accuracy: 0.3077 - val_loss: 1.6219
Epoch 2/10
24/24             36s 51ms/step -
accuracy: 0.3283 - loss: 1.6785 - val_accuracy: 0.3462 - val_loss: 1.4098
Epoch 3/10
24/24             2s 44ms/step -
accuracy: 0.4006 - loss: 1.3737 - val_accuracy: 0.4615 - val_loss: 1.2993
```

```
Epoch 4/10
24/24             1s 44ms/step -
accuracy: 0.4219 - loss: 1.3380 - val_accuracy: 0.5000 - val_loss: 1.2318
Epoch 5/10
24/24             1s 42ms/step -
accuracy: 0.3794 - loss: 1.3442 - val_accuracy: 0.5385 - val_loss: 1.1700
Epoch 6/10
24/24             1s 43ms/step -
accuracy: 0.5160 - loss: 1.1333 - val_accuracy: 0.5000 - val_loss: 1.1155
Epoch 7/10
24/24             1s 42ms/step -
accuracy: 0.5342 - loss: 0.9801 - val_accuracy: 0.5000 - val_loss: 1.0714
Epoch 8/10
24/24             1s 44ms/step -
accuracy: 0.5352 - loss: 1.0766 - val_accuracy: 0.5000 - val_loss: 1.0460
Epoch 9/10
24/24             1s 45ms/step -
accuracy: 0.6446 - loss: 0.9201 - val_accuracy: 0.5769 - val_loss: 1.0085
Epoch 10/10
24/24             1s 44ms/step -
accuracy: 0.5975 - loss: 0.9222 - val_accuracy: 0.5769 - val_loss: 0.9684
```

[ ]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']
```

[ ]:
```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,2.5])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



La curva de pérdida del conjunto de validación disminuye su valor con una pendiente relativamente suave, pero la curva de pérdida del conjunto de entrenamiento presenta quiebres en su tendencia.

La curva de accuracy del conjunto de validación fluctúa, mostrando inestabilidad en sus valores.

[ ]:
```
# Permite entrenamiento del modelo base
base_model_s.trainable = True
```

[ ]:
```
# Verifica cantidad de capas del modelo base
print("Number of layers in the base model: ", len(base_model_s.layers))

# Fine-tune  desde esta capa
fine_tune_at = 189
```

```
# Congela capas anteriores a la capa desde la que se realizará el fine-tuning
for layer in base_model_s.layers[:fine_tune_at]:
  layer.trainable = False
```

```
Number of layers in the base model:  190
```

[ ]:
```
model_s.compile(loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=False,
              ignore_class=None, reduction='sum_over_batch_size',␣
 ↪name='sparse_categorical_crossentropy'),
              optimizer = tf.keras.optimizers.
 ↪RMSprop(learning_rate=base_learning_rate/10),
              metrics = ['accuracy'])
```

[ ]:
```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs
```

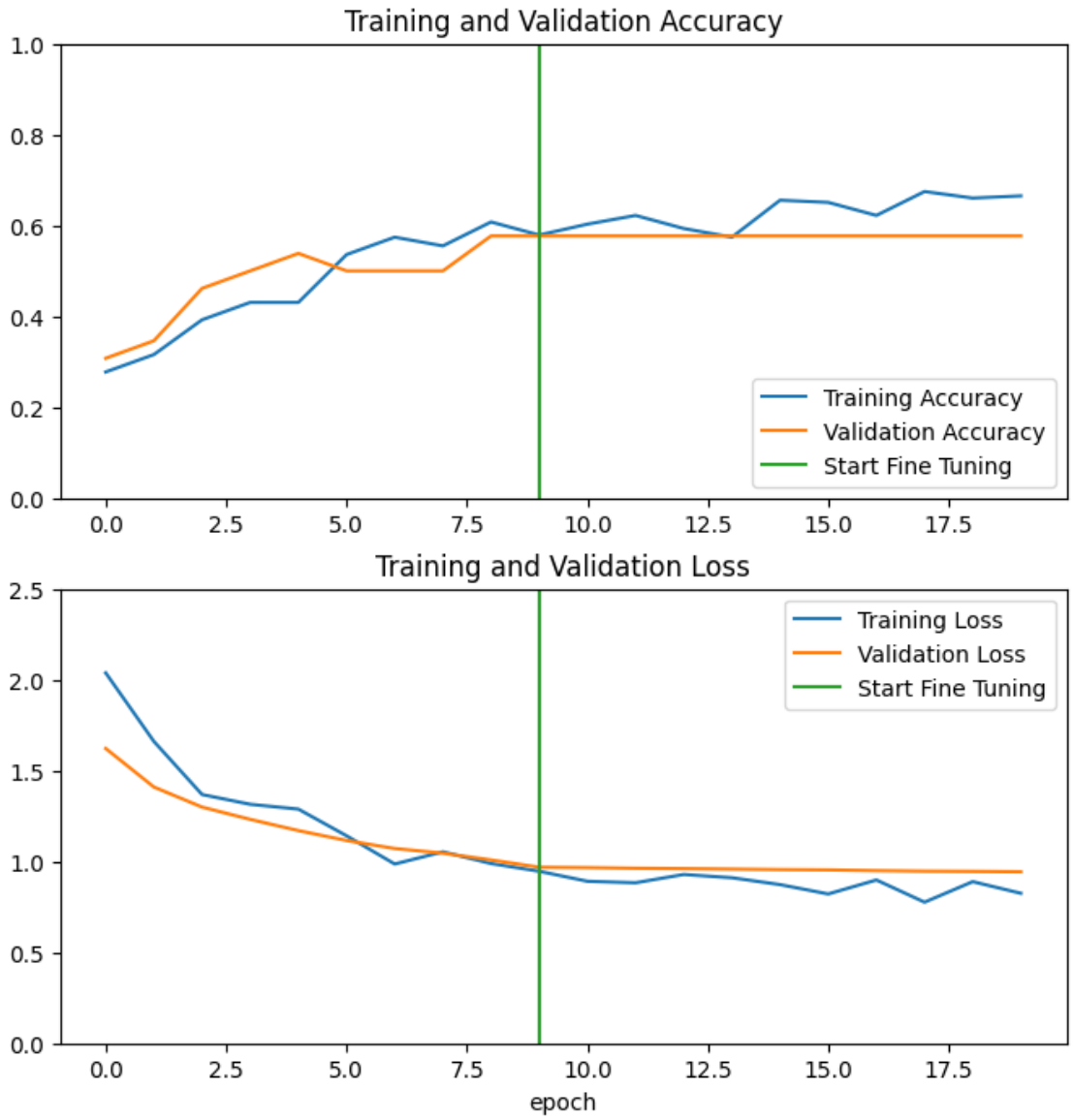
[ ]:
```
history_fine = model_s.fit(train_dataset_s,
                         epochs=total_epochs,
                         initial_epoch=len(history.epoch),
                         validation_data=validation_dataset_s,
                         callbacks=[save_checkpoint_3])
```

```
Epoch 11/20
23/24             0s 39ms/step -
accuracy: 0.6304 - loss: 0.8780
Epoch 11: val_loss improved from inf to 0.96624, saving model to
./models/model_s_ResNet50V2.keras
24/24             12s 165ms/step -
accuracy: 0.6282 - loss: 0.8791 - val_accuracy: 0.5769 - val_loss: 0.9662
Epoch 12/20
24/24             0s 40ms/step -
accuracy: 0.5556 - loss: 0.9281
Epoch 12: val_loss improved from 0.96624 to 0.96263, saving model to
./models/model_s_ResNet50V2.keras
24/24             2s 86ms/step -
accuracy: 0.5582 - loss: 0.9263 - val_accuracy: 0.5769 - val_loss: 0.9626
Epoch 13/20
24/24             0s 41ms/step -
accuracy: 0.5897 - loss: 0.9060
Epoch 13: val_loss improved from 0.96263 to 0.96079, saving model to
./models/model_s_ResNet50V2.keras
24/24             2s 88ms/step -
accuracy: 0.5898 - loss: 0.9069 - val_accuracy: 0.5769 - val_loss: 0.9608
Epoch 14/20
23/24             0s 38ms/step -
accuracy: 0.6196 - loss: 0.8231
```

```
Epoch 14: val_loss improved from 0.96079 to 0.95820, saving model to
./models/model_s_ResNet50V2.keras
24/24            2s 75ms/step -
accuracy: 0.6160 - loss: 0.8301 - val_accuracy: 0.5769 - val_loss: 0.9582
Epoch 15/20
23/24            0s 38ms/step -
accuracy: 0.6507 - loss: 0.8792
Epoch 15: val_loss improved from 0.95820 to 0.95545, saving model to
./models/model_s_ResNet50V2.keras
24/24            2s 76ms/step -
accuracy: 0.6511 - loss: 0.8786 - val_accuracy: 0.5769 - val_loss: 0.9554
Epoch 16/20
24/24            0s 49ms/step -
accuracy: 0.6201 - loss: 0.8197
Epoch 16: val_loss improved from 0.95545 to 0.95379, saving model to
./models/model_s_ResNet50V2.keras
24/24            2s 87ms/step -
accuracy: 0.6213 - loss: 0.8197 - val_accuracy: 0.5769 - val_loss: 0.9538
Epoch 17/20
24/24            0s 50ms/step -
accuracy: 0.6158 - loss: 0.9361
Epoch 17: val_loss improved from 0.95379 to 0.94981, saving model to
./models/model_s_ResNet50V2.keras
24/24            3s 89ms/step -
accuracy: 0.6160 - loss: 0.9346 - val_accuracy: 0.5769 - val_loss: 0.9498
Epoch 18/20
24/24            0s 52ms/step -
accuracy: 0.7054 - loss: 0.7239
Epoch 18: val_loss improved from 0.94981 to 0.94685, saving model to
./models/model_s_ResNet50V2.keras
24/24            3s 102ms/step -
accuracy: 0.7042 - loss: 0.7260 - val_accuracy: 0.5769 - val_loss: 0.9468
Epoch 19/20
24/24            0s 53ms/step -
accuracy: 0.6410 - loss: 0.9171
Epoch 19: val_loss improved from 0.94685 to 0.94572, saving model to
./models/model_s_ResNet50V2.keras
24/24            2s 92ms/step -
accuracy: 0.6418 - loss: 0.9160 - val_accuracy: 0.5769 - val_loss: 0.9457
Epoch 20/20
24/24            0s 50ms/step -
accuracy: 0.6638 - loss: 0.8088
Epoch 20: val_loss improved from 0.94572 to 0.94379, saving model to
./models/model_s_ResNet50V2.keras
24/24            2s 87ms/step -
accuracy: 0.6639 - loss: 0.8094 - val_accuracy: 0.5769 - val_loss: 0.9438
```

[ ]:
```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']
```

[ ]:
```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0, 1])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 2.5])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Durante el proceso de fine-tuning, las curvas de accuracy y loss del conuunto de validación se mantienen estables. Sin embargo, la curva de pérdida del conjunto de entrenamiento presenta fluctuaciones.

[ ]:
```
reconstructed_model_3 = tf.keras.models.load_model('./models/model_s_ResNet50V2.
↪keras')
```

[ ]:
```
score3 = reconstructed_model_3.evaluate(validation_dataset_s, verbose=False)
print('Val loss:', score3[0])
print('Val accuracy:', score3[1])
```

```
Val loss: 0.9437882900238037
```

```
Val accuracy: 0.5769230723381042
```

```
[ ]: score3 = reconstructed_model_3.evaluate(test_dataset_s, verbose=False)
print('Test loss:', score3[0])
print('Test accuracy:', score3[1])
```

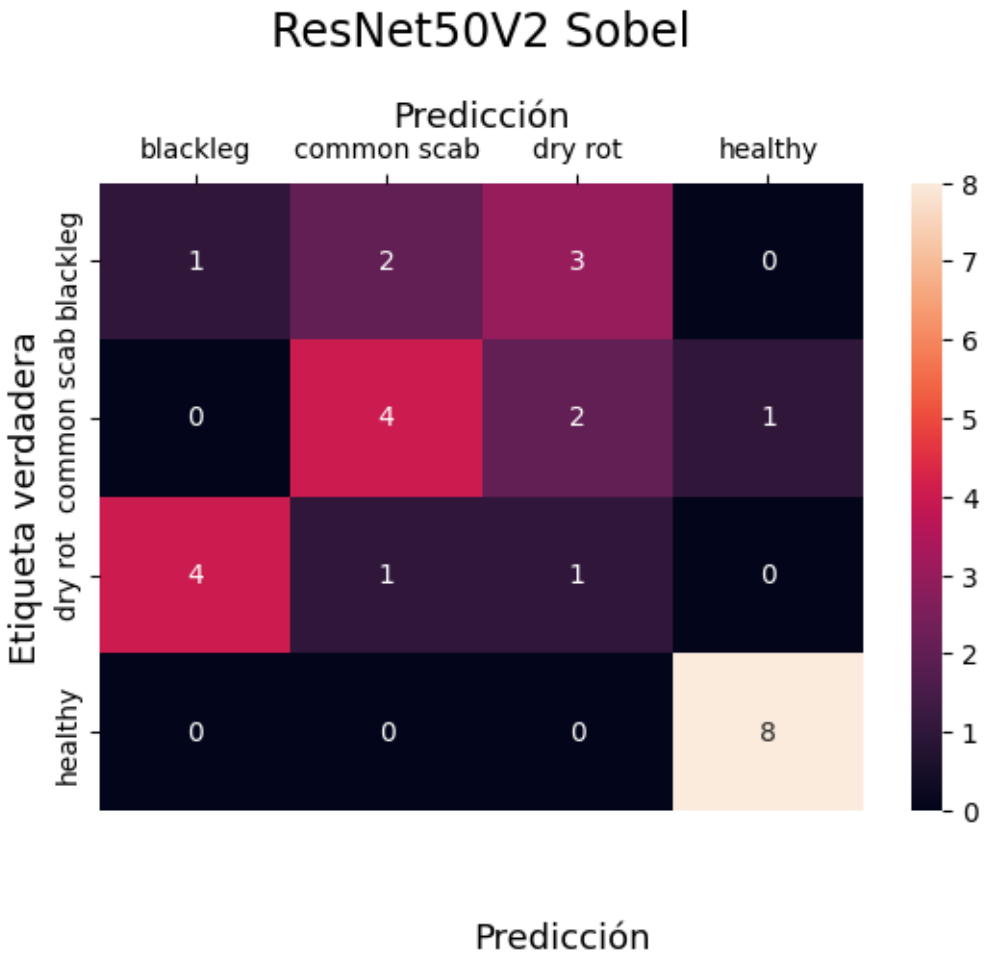
```
Test loss: 1.0531729459762573
Test accuracy: 0.5185185074806213
```

Los valores de pérdida de los conjuntos de entrenamiento son muy altos, lo cual implica que el modelo no aprendió realmente a distinguir las cuatro clases de papas del dataset. Los resultados obtenidos son prácticamente aleatorios en este caso.

3.6.1 Matriz de confusión

```
[ ]: def get_true_labels_and_predictions(model, dataset):
    true_labels = []
    predictions = []
    for images, labels in dataset:
        true_labels.extend(labels.numpy())
        preds = model.predict(images)
        predictions.extend(np.argmax(preds, axis=1))
    return np.array(true_labels), np.array(predictions)

y_test_3, y_pred_3 = get_true_labels_and_predictions(reconstructed_model_3,␣
 ↪test_dataset_s)
cm = confusion_matrix(y_test_3, y_pred_3)
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['blackleg', 'common scab', 'dry rot', 'healthy'],
            yticklabels=['blackleg', 'common scab', 'dry rot', 'healthy'])

plt.ylabel('Etiqueta verdadera', fontsize=13)
plt.title('ResNet50V2 Sobel', fontsize=17, pad=20)
plt.gca().xaxis.set_label_position('top')
plt.xlabel('Predicción', fontsize=13)
plt.gca().xaxis.tick_top()

plt.gca().figure.subplots_adjust(bottom=0.2)
plt.gca().figure.text(0.5, 0.05, 'Predicción', ha='center', fontsize=13)
plt.show()
```

```
1/1             2s 2s/step
1/1             0s 186ms/step
1/1             0s 83ms/step
```



Los resultados presentados en la matriz de confusión corroboran en mal desempeño del modelo, debido a que solamente una de seis papas con blackleg fue clasificada correctamente.

Cuatro papas fueron clasificadas como papas infectadas por common scab de forma correcta, pero dos fueron etiquetadas erróneamente como papas infectadas por dry rot y una fue clasificada como una papa sana.

Cuatro papas que están infectadas por dry rot fueron etiquetadas como infectadas por common scab, y una fue etiquetada como una papa sana. Es altamente probable que la detección de bordes del filtro Sobel cause esta confusión, debido a que no están presentes los colores distintivos de la enfermedad blackleg y dry rot.

Todas las papas sanas fueron etiquetadas correctamente.

[ ]:
```
def c_report(y_true, y_pred, class_names):
    report = classification_report(y_true, y_pred, target_names=class_names,␣
 ↪zero_division=1)
    return report
```

```
report_3 = c_report(y_test_3, y_pred_3, class_names_s)

print(f"Classification Report for Model 3:\n{report_3}")

El modelo en este caso distingue bien a las papas sanas de las papas infectadas␣
 ↪por alguna enfermedad. Sin embargo, no es capaz de determinar de buena␣
 ↪manera si una papa está infectada por blackleg o por dry rot, dado que sus␣
 ↪métricas son las peores del dataset.
```

3.6.2 Determinación del tiempo de ejecución del proceso de fine-tuning del modelo ResNet50V2 para el 50% y el 100% de las imágenes del dataset

El tiempo de procesamiento del 50% y 100% de las imágenes del dataset considera el uso de una tarjeta Nvidia T4 para acelerarlo. Sólo se mide cinco veces debido a que la GPU es un recurso costoso de utilizar.

[ ]:
```
ex_time = []
half_time = []

j = 1

while j < 6:
    start = time.perf_counter()


    fine_tune_epochs = 10
    total_epochs =  initial_epochs + fine_tune_epochs

    history_fine = model_s.fit(train_dataset_s,
                         epochs=total_epochs,
                         initial_epoch=len(history.epoch),
                         validation_data=validation_dataset_s)

    end = time.perf_counter()
    total = end - start
    print(f"Tiempo transcurrido: {(total)} s")
    ex_time.append(total)
    half_t = (total/2)
    half_time.append(half_t)
    j +=1
```

[ ]:
```
print("Tiempo mínimo ", min(ex_time), 's')
print("Tiempo máximo: ", max(ex_time), 's')
print("Tiempo promedio: ", sum(ex_time)/len(ex_time),'s')
print("Desviación estándar del tiempo: ", np.std(ex_time), 's')
```

```
Tiempo mínimo  11.523368169999912 s
Tiempo máximo:  12.933443750999913 s
Tiempo promedio:  12.03809938039999 s
Desviación estándar del tiempo:  0.5190170317114993 s
```

[ ]:
```
print("Tiempo mínimo mitad ejecución ", min(half_time), 's')
print("Tiempo máximo mitad ejecución: ", max(half_time), 's')
print("Tiempo promedio mitad ejecución: ", sum(half_time)/len(half_time), 's')
print("Desviación estándar del tiempo mitad ejecución: ", np.std(half_time),␣
 ↪'s')
```

```
Tiempo mínimo mitad ejecución  5.761684084999956 s
Tiempo máximo mitad ejecución:  6.466721875499957 s
Tiempo promedio mitad ejecución:  6.019049690199995 s
Desviación estándar del tiempo mitad ejecución:  0.25950851585574963 s
```

3.7 Fine-tuning modelo ResNet50V2 para dataset filtro Máximo

Solamente se realizó una vez el paso de dividir las carpetas en train, validation y test. En pruebas posteriores se comentó esa parte del código debido a que las carpetas fueron almacenadas en Google Drive.

[ ]:
```
# Dividir carpeta en train, validation y test

path_model = "/content/drive/MyDrive/potato/data_model_max/" # Carpeta para␣
 ↪guardar imágenes
# Crea la carpeta de salida si no existe
if not os.path.exists(path_model):
    os.makedirs(path_model)

splitfolders.ratio("/content/drive/MyDrive/potato/data_max/", output="/content/
 ↪drive/MyDrive/potato/data_model_max/",
    seed=42, ratio=(.8, .1, .1), group_prefix=None, move=False) # default values
```

[ ]:
```
train_dir_max = os.path.join(path_model, 'train')
validation_dir_max = os.path.join(path_model, 'val')
test_dir_max = os.path.join(path_model,'test')

BATCH_SIZE = 9
IMG_SIZE = (200, 200)

train_dataset_max = tf.keras.utils.image_dataset_from_directory(train_dir_max,
                                                                shuffle=True,
                                                                ␣
 ↪batch_size=BATCH_SIZE,
                                                                image_size=IMG_SIZE)
```

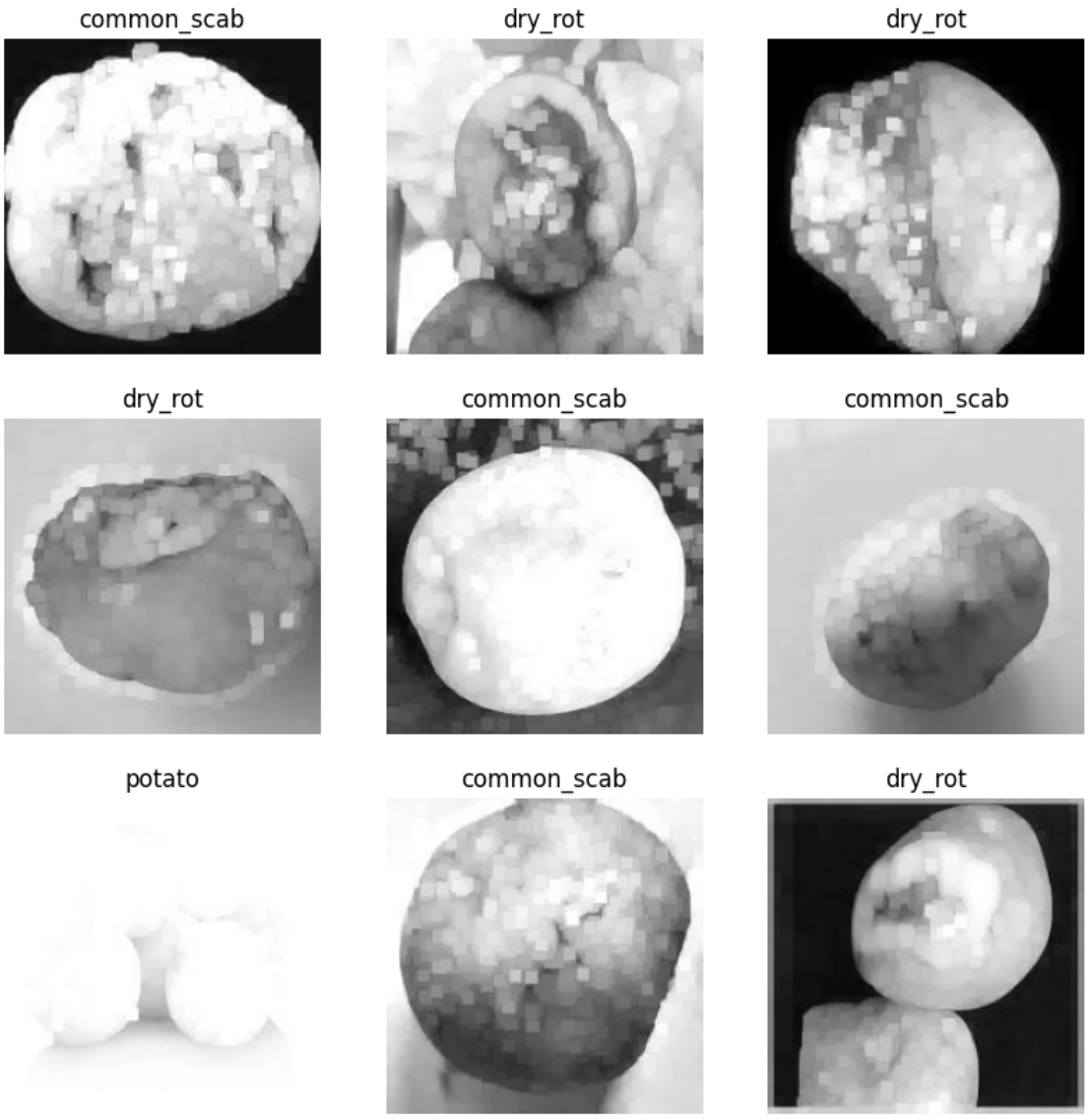
```
Found 209 files belonging to 4 classes.
```

[ ]:
```
validation_dataset_max = tf.keras.utils.
  ↪image_dataset_from_directory(validation_dir_max,
                                                                  shuffle=True,
                                                                ␣
  ↪batch_size=BATCH_SIZE,
                                                                ␣
  ↪image_size=IMG_SIZE)

test_dataset_max = tf.keras.utils.image_dataset_from_directory(test_dir_max,
                                                                  shuffle=True,
                                                                ␣
  ↪batch_size=BATCH_SIZE,
                                                                ␣
  ↪image_size=IMG_SIZE)
```

```
Found 26 files belonging to 4 classes.
Found 27 files belonging to 4 classes.
```

Se visualiza una muestra de las imágenes del conjunto de entrenamiento y sus respectivas etiquetas en una cuadrícula de 3x3.

[ ]:
```
class_names_max = train_dataset_max.class_names

plt.figure(figsize=(10, 10))
for images, labels in train_dataset_max.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names_max[labels[i]])
    plt.axis("off")
```



Se verifica que el ejemplo de papa sana mostrado en esta muestra apenas se distingue del fondo blanco, lo cual afectaría negativamente la calidad de los resultados del proceso de fine-tuning.

[ ]:
```
# Ajusta parámetros de memoria automáticamente a la capacidad del computador
AUTOTUNE = tf.data.AUTOTUNE

train_dataset_max = train_dataset_max.prefetch(buffer_size=AUTOTUNE)
validation_dataset_max = validation_dataset_max.prefetch(buffer_size=AUTOTUNE)
test_dataset_max = test_dataset_max.prefetch(buffer_size=AUTOTUNE)
```

[ ]:
```
# Aplica data augmentation al dataset de papas
```

```
data_augmentation_max = tf.keras.Sequential([
  tf.keras.layers.RandomFlip('horizontal'),
  tf.keras.layers.RandomRotation(0.1),
  tf.keras.layers.RandomTranslation(height_factor=0.1, width_factor=0.1),
  tf.keras.layers.RandomContrast(factor=0.1),
])
```

[ ]:
```
# Define el modelo que será aplicado para transformar las imágenes en vectores
preprocess_input = tf.keras.applications.resnet_v2.preprocess_input
```

[ ]:
```
# Crea el modelo base a partir del modelo pre-entrenado a partir del modelo␣
 ↪ResNet50V2
IMG_SHAPE = IMG_SIZE + (3,)
base_model_max = tf.keras.applications.ResNet50V2(input_shape=IMG_SHAPE,
                                                  include_top=False,
                                                  weights='imagenet')
```

En este paso se convierte a las imágenes en bloques de características de tamaño 7x7x2048.

[ ]:
```
image_batch, label_batch = next(iter(train_dataset_max))
feature_batch = base_model_max(image_batch)
print(feature_batch.shape)
```

```
(9, 7, 7, 2048)
```

[ ]:
```
base_model_max.trainable = False
```

[ ]:
```
# Agrega capa de clasificación para transformar las características en vectores␣
 ↪de 2048 elementos
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(9, 2048)
```

[ ]:
```
# Capa de predicción para transformar las características en una sola␣
 ↪predicción por imagen
num_classes = len(class_names_max)
prediction_layer = tf.keras.layers.Dense(num_classes, activation='softmax')
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(9, 4)
```

[ ]:
```
# Concatena data augmentation, preprocesamiento, el modelo base, y capas que␣
 ↪extraen características.

inputs = tf.keras.Input(shape=(200, 200, 3))
x = data_augmentation_g(inputs)
```

```
x = preprocess_input(x)
x = base_model_max(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model_max = tf.keras.Model(inputs, outputs)
```

[ ]:
```
base_learning_rate = 0.0001
model_max.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=False,
              ignore_class=None, reduction='sum_over_batch_size',␣
 ↪name='sparse_categorical_crossentropy'),
              metrics= ['accuracy'])
```

[ ]:
```
initial_epochs = 10

loss0, accuracy0 = model_max.evaluate(validation_dataset_max)
```

```
3/3             12s 3s/step -
accuracy: 0.2350 - loss: 1.8205
```

[ ]:
```
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
initial loss: 1.87
initial accuracy: 0.19
```

[ ]:
```
# guardar el modelo durante el entrenamiento de la red neuronal.
save_checkpoint_4 = tf.keras.callbacks.ModelCheckpoint(
    filepath='./models/model_max_ResNet50V2.keras',␣
 ↪monitor='val_loss',mode='min', save_best_only=True, verbose=1
    )
```

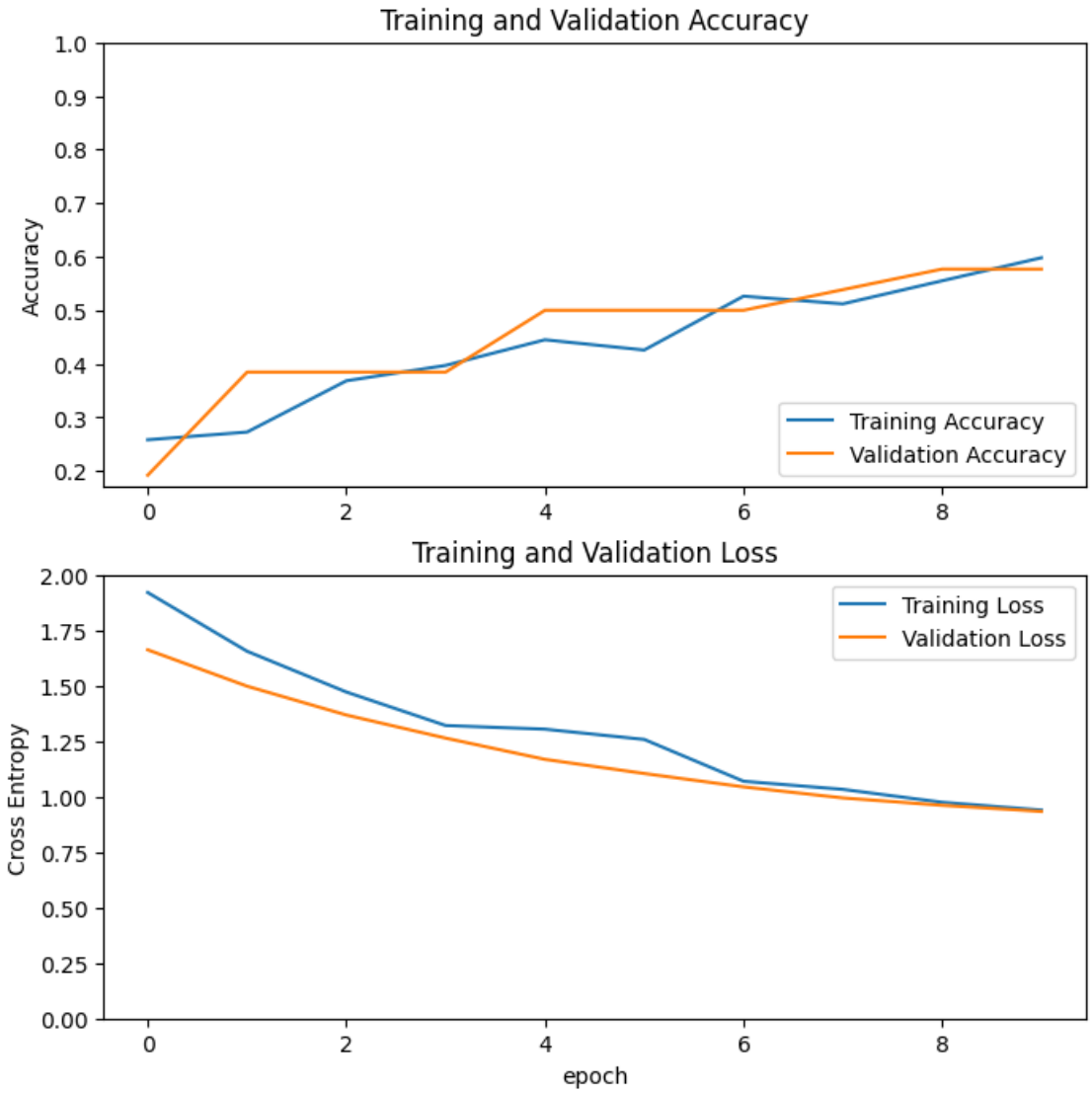
[ ]:
```
history = model_max.fit(train_dataset_max,
                    epochs=initial_epochs,
                    validation_data=validation_dataset_max)
```

```
Epoch 1/10
24/24             53s 2s/step -
accuracy: 0.2472 - loss: 1.9401 - val_accuracy: 0.1923 - val_loss: 1.6617
Epoch 2/10
24/24             37s 47ms/step -
accuracy: 0.2442 - loss: 1.6531 - val_accuracy: 0.3846 - val_loss: 1.4977
Epoch 3/10
24/24             1s 44ms/step -
accuracy: 0.3774 - loss: 1.5001 - val_accuracy: 0.3846 - val_loss: 1.3678
Epoch 4/10
```

```
24/24              1s 45ms/step -
accuracy: 0.3443 - loss: 1.4363 - val_accuracy: 0.3846 - val_loss: 1.2637
Epoch 5/10
24/24              1s 44ms/step -
accuracy: 0.4063 - loss: 1.3785 - val_accuracy: 0.5000 - val_loss: 1.1680
Epoch 6/10
24/24              1s 45ms/step -
accuracy: 0.3738 - loss: 1.3703 - val_accuracy: 0.5000 - val_loss: 1.1042
Epoch 7/10
24/24              2s 57ms/step -
accuracy: 0.5117 - loss: 1.1435 - val_accuracy: 0.5000 - val_loss: 1.0438
Epoch 8/10
24/24              2s 43ms/step -
accuracy: 0.4367 - loss: 1.1183 - val_accuracy: 0.5385 - val_loss: 0.9941
Epoch 9/10
24/24              1s 43ms/step -
accuracy: 0.4709 - loss: 1.0455 - val_accuracy: 0.5769 - val_loss: 0.9608
Epoch 10/10
24/24              1s 43ms/step -
accuracy: 0.5558 - loss: 1.0077 - val_accuracy: 0.5769 - val_loss: 0.9334
```

[ ]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']
```

[ ]:
```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,2.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



La curva de pérdida del conjunto de validación no presenta ruido, mientras que la curva de pérdida del conjunto de entrenamiento presenta quiebres en su tendencia.

En el caso de las curvas de accuracy se observan muchas fluctuaciones, lo cual implica que el modelo no estaría funcionando adecuadamente.

[ ]:
```
# Permite entrenamiento del modelo base
base_model_max.trainable = True
```

[ ]:
```
# Verifica cantidad de capas del modelo base
print("Number of layers in the base model: ", len(base_model_max.layers))

# Fine-tune  desde esta capa
fine_tune_at = 189
```

```
# Congela capas anteriores a la capa desde la que se realizará el fine-tuning
for layer in base_model_max.layers[:fine_tune_at]:
  layer.trainable = False
```

```
Number of layers in the base model:  190
```

[ ]:
```
model_max.compile(loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=False,
              ignore_class=None, reduction='sum_over_batch_size',␣
 ↪name='sparse_categorical_crossentropy'),
              optimizer = tf.keras.optimizers.
 ↪RMSprop(learning_rate=base_learning_rate/10),
              metrics = ['accuracy'])
```

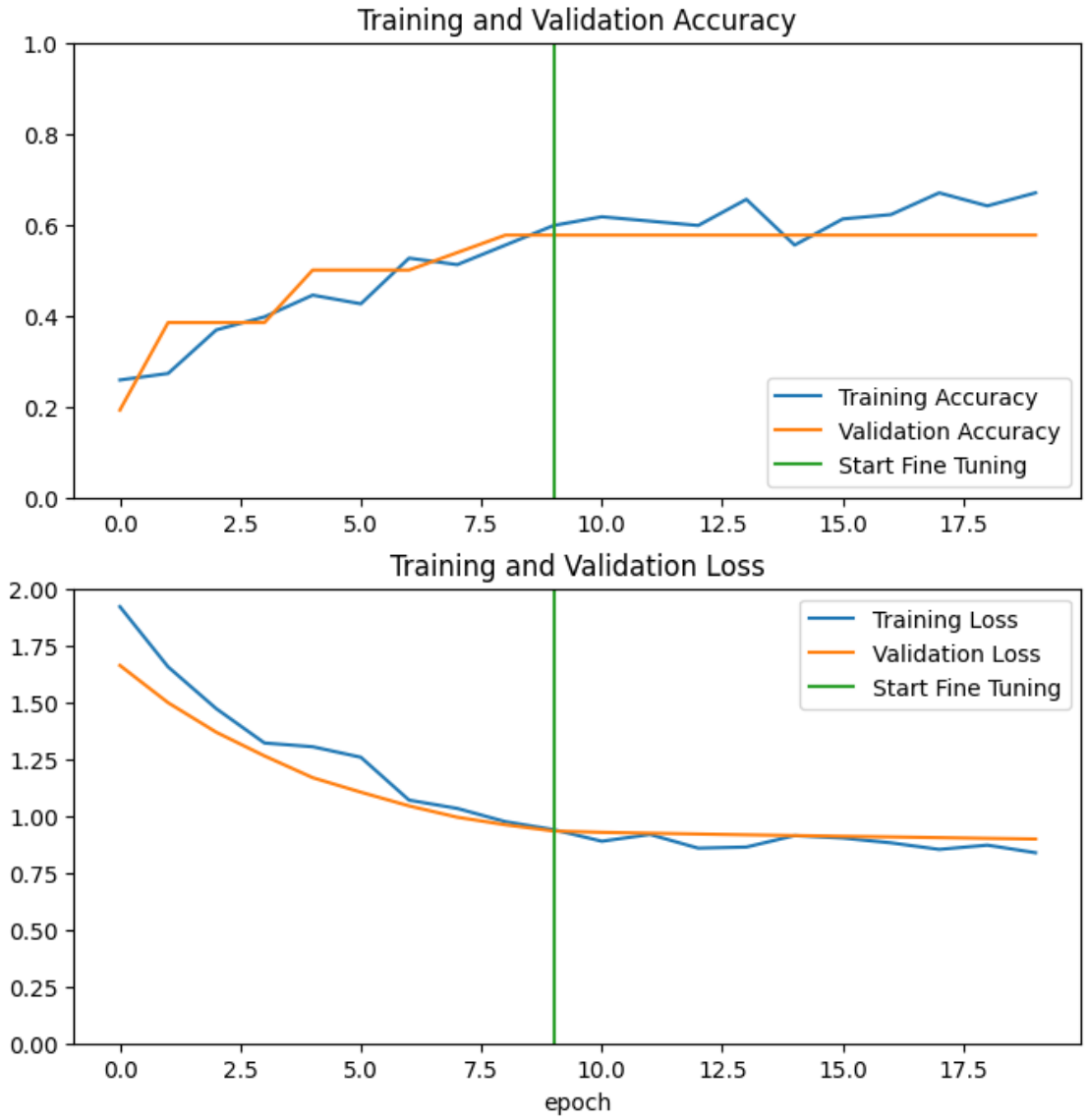
[ ]:
```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model_max.fit(train_dataset_max,
                         epochs=total_epochs,
                         initial_epoch=len(history.epoch),
                         validation_data=validation_dataset_max,
                         callbacks=[save_checkpoint_4])
```

```
Epoch 11/20
24/24             0s 40ms/step -
accuracy: 0.6248 - loss: 0.9261
Epoch 11: val_loss improved from inf to 0.92741, saving model to
./models/model_max_ResNet50V2.keras
24/24             12s 167ms/step -
accuracy: 0.6245 - loss: 0.9246 - val_accuracy: 0.5769 - val_loss: 0.9274
Epoch 12/20
23/24             0s 49ms/step -
accuracy: 0.5543 - loss: 0.9434
Epoch 12: val_loss improved from 0.92741 to 0.92379, saving model to
./models/model_max_ResNet50V2.keras
24/24             3s 97ms/step -
accuracy: 0.5586 - loss: 0.9414 - val_accuracy: 0.5769 - val_loss: 0.9238
Epoch 13/20
23/24             0s 55ms/step -
accuracy: 0.5386 - loss: 0.9167
Epoch 13: val_loss improved from 0.92379 to 0.92011, saving model to
./models/model_max_ResNet50V2.keras
24/24             2s 94ms/step -
accuracy: 0.5434 - loss: 0.9120 - val_accuracy: 0.5769 - val_loss: 0.9201
Epoch 14/20
23/24             0s 50ms/step -
accuracy: 0.6268 - loss: 0.9343
```

```
Epoch 14: val_loss improved from 0.92011 to 0.91670, saving model to
./models/model_max_ResNet50V2.keras
24/24            2s 87ms/step -
accuracy: 0.6291 - loss: 0.9285 - val_accuracy: 0.5769 - val_loss: 0.9167
Epoch 15/20
23/24            0s 37ms/step -
accuracy: 0.5609 - loss: 0.9336
Epoch 15: val_loss improved from 0.91670 to 0.91346, saving model to
./models/model_max_ResNet50V2.keras
24/24            3s 74ms/step -
accuracy: 0.5604 - loss: 0.9320 - val_accuracy: 0.5769 - val_loss: 0.9135
Epoch 16/20
23/24            0s 47ms/step -
accuracy: 0.5971 - loss: 0.9128
Epoch 16: val_loss improved from 0.91346 to 0.91010, saving model to
./models/model_max_ResNet50V2.keras
24/24            2s 85ms/step -
accuracy: 0.5983 - loss: 0.9119 - val_accuracy: 0.5769 - val_loss: 0.9101
Epoch 17/20
22/24            0s 50ms/step -
accuracy: 0.6021 - loss: 0.8997
Epoch 17: val_loss improved from 0.91010 to 0.90741, saving model to
./models/model_max_ResNet50V2.keras
24/24            2s 85ms/step -
accuracy: 0.6045 - loss: 0.8975 - val_accuracy: 0.5769 - val_loss: 0.9074
Epoch 18/20
23/24            0s 61ms/step -
accuracy: 0.6361 - loss: 0.8809
Epoch 18: val_loss improved from 0.90741 to 0.90422, saving model to
./models/model_max_ResNet50V2.keras
24/24            3s 110ms/step -
accuracy: 0.6388 - loss: 0.8786 - val_accuracy: 0.5769 - val_loss: 0.9042
Epoch 19/20
23/24            0s 37ms/step -
accuracy: 0.6056 - loss: 0.9068
Epoch 19: val_loss improved from 0.90422 to 0.90101, saving model to
./models/model_max_ResNet50V2.keras
24/24            2s 76ms/step -
accuracy: 0.6084 - loss: 0.9040 - val_accuracy: 0.5769 - val_loss: 0.9010
Epoch 20/20
23/24            0s 49ms/step -
accuracy: 0.6112 - loss: 0.9065
Epoch 20: val_loss improved from 0.90101 to 0.89823, saving model to
./models/model_max_ResNet50V2.keras
24/24            3s 100ms/step -
accuracy: 0.6159 - loss: 0.9010 - val_accuracy: 0.5769 - val_loss: 0.8982
```

[ ]:
```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']
```

[ ]:
```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0, 1])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 2.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Durante el proceso de fine-tuning se mantienen estables las curvas correspondientes al conjunto de validación, pero se observan fluctuaciones en las curvas del conjunto de entrenamiento.

[ ]:
```
reconstructed_model_4 = tf.keras.models.load_model('./models/
  ↪model_max_ResNet50V2.keras')
```

[ ]:
```
score4 = reconstructed_model_4.evaluate(validation_dataset_max, verbose=False)
print('Val loss:', score4[0])
print('Val accuracy:', score4[1])

score4 = reconstructed_model_4.evaluate(test_dataset_max, verbose=False)
print('Test loss:', score4[0])
```

```
print('Test accuracy:', score4[1])
```

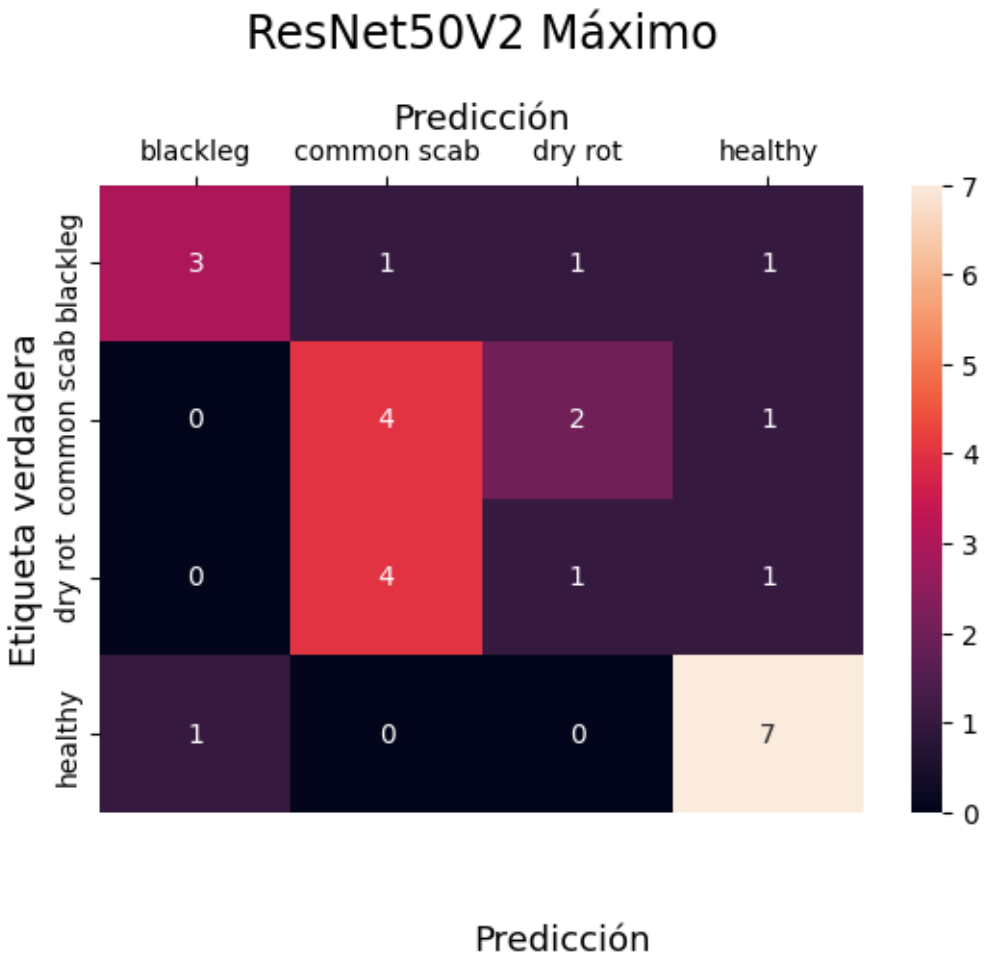
```
Val loss: 0.8982301950454712
Val accuracy: 0.5769230723381042
Test loss: 0.9364304542541504
Test accuracy: 0.5555555820465088
```

Los valores de pérdida de los conjuntos de validación y prueba son muy altos, por lo tanto, los resultados obtenidos probablemente se están obteniendo al azar.

3.7.1 Matriz de confusión y métricas

[ ]:
```
def get_true_labels_and_predictions(model, dataset):
    true_labels = []
    predictions = []
    for images, labels in dataset:
        true_labels.extend(labels.numpy())
        preds = model.predict(images)
        predictions.extend(np.argmax(preds, axis=1))
    return np.array(true_labels), np.array(predictions)

y_test_4, y_pred_4 = get_true_labels_and_predictions(reconstructed_model_4,␣
 ↪test_dataset_max)
cm = confusion_matrix(y_test_4, y_pred_4)
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['blackleg', 'common scab', 'dry rot', 'healthy'],
            yticklabels=['blackleg', 'common scab', 'dry rot', 'healthy'])

plt.ylabel('Etiqueta verdadera', fontsize=13)
plt.title('ResNet50V2 Máximo', fontsize=17, pad=20)
plt.gca().xaxis.set_label_position('top')
plt.xlabel('Predicción', fontsize=13)
plt.gca().xaxis.tick_top()

plt.gca().figure.subplots_adjust(bottom=0.2)
plt.gca().figure.text(0.5, 0.05, 'Predicción', ha='center', fontsize=13)
plt.show()
```

```
1/1             3s 3s/step
1/1             0s 84ms/step
1/1             0s 75ms/step
```



La mitad de las papas con blackleg fueron etiquetadas incorrectamente, siendo una clasificada como infectada por common scab, una como infectada por dry rot y otra como una papa sana.

Tres papas con common scab fueron clasificadas erróneamente, siendo dos etiquetadas como papas infectadas por dry rot y una como una papa sana.

Cuatro papas infectadas por dry rot fueron clasificadas como papas infectadas por common scab, y una como papa sana.

Una papa sana fue clasificada como papa infectada por blackleg.

[ ]:
```
def c_report(y_true, y_pred, class_names):
    report = classification_report(y_true, y_pred, target_names=class_names,␣
 ↪zero_division=1)
    return report

report_4 = c_report(y_test_4, y_pred_4, class_names_max)
```

```
print(f"Classification Report for Model 4:\n{report_4}")
```

```
Classification Report for Model 4:
              precision    recall  f1-score   support

    blackleg       0.75      0.50      0.60         6
 common_scab       0.44      0.57      0.50         7
     dry_rot       0.25      0.17      0.20         6
      potato       0.70      0.88      0.78         8

    accuracy                           0.56        27
   macro avg       0.54      0.53      0.52        27
weighted avg       0.54      0.56      0.54        27
```

En general, los resultados son mejores para la clase potato, fallando demasiado en el caso de la clase dry rot. Es muy probable que el aclaramiento de las imágenes impida distinguir la clase dry rot de la clase common scab.

3.7.2 Determinación del tiempo de ejecución del proceso de fine-tuning del modelo ResNet50V2 para el 50% y el 100% de las imágenes del dataset

El tiempo de procesamiento del 50% y 100% de las imágenes del dataset considera el uso de una tarjeta Nvidia T4 para acelerarlo. Sólo se mide cinco veces debido a que la GPU es un recurso costoso de utilizar.

[ ]:
```
ex_time = []
half_time = []

j = 1

while j < 6:
    start = time.perf_counter()


    fine_tune_epochs = 10
    total_epochs =  initial_epochs + fine_tune_epochs

    history_fine = model_max.fit(train_dataset_max,
                         epochs=total_epochs,
                         initial_epoch=len(history.epoch),
                         validation_data=validation_dataset_max)

    end = time.perf_counter()
    total = end - start
    print(f"Tiempo transcurrido: {(total)} s")
    ex_time.append(total)
    half_t = (total/2)
```

```
    half_time.append(half_t)
    j +=1
```

[ ]:
```
print("Tiempo mínimo ", min(ex_time), 's')
print("Tiempo máximo: ", max(ex_time), 's')
print("Tiempo promedio: ", sum(ex_time)/len(ex_time),'s')
print("Desviación estándar del tiempo: ", np.std(ex_time), 's')
```

```
Tiempo mínimo  11.446188221999819 s
Tiempo máximo:  14.352345911999691 s
Tiempo promedio:  12.639684457599833 s
Desviación estándar del tiempo:  1.1702517020418268 s
```

[ ]:
```
print("Tiempo mínimo mitad ejecución ", min(half_time), 's')
print("Tiempo máximo mitad ejecución: ", max(half_time), 's')
print("Tiempo promedio mitad ejecución: ", sum(half_time)/len(half_time), 's')
print("Desviación estándar del tiempo mitad ejecución: ", np.std(half_time),␣
 ↪'s')
```

```
Tiempo mínimo mitad ejecución  5.723094110999909 s
Tiempo máximo mitad ejecución:  7.176172955999846 s
Tiempo promedio mitad ejecución:  6.319842228799916 s
Desviación estándar del tiempo mitad ejecución:  0.5851258510209134 s
```

3.8 Fine-tuning modelo ResNet50V2 para dataset filtro Mínimo

Solamente se realizó una vez el paso de dividir las carpetas en train, validation y test. En pruebas posteriores se comentó esa parte del código debido a que las carpetas fueron almacenadas en Google Drive.

[ ]:
```
# Dividir carpeta en train, validation y test

path_model = "/content/drive/MyDrive/potato/data_model_min/" # Carpeta para␣
 ↪guardar imágenes
# Crea la carpeta de salida si no existe
if not os.path.exists(path_model):
    os.makedirs(path_model)

splitfolders.ratio("/content/drive/MyDrive/potato/data_min/", output="/content/
 ↪drive/MyDrive/potato/data_model_min/",
    seed=42, ratio=(.8, .1, .1), group_prefix=None, move=False) # default values
```

[ ]:
```
train_dir_min = os.path.join(path_model, 'train')
validation_dir_min = os.path.join(path_model, 'val')
test_dir_min = os.path.join(path_model,'test')

BATCH_SIZE = 9
IMG_SIZE = (200, 200)
```

```
train_dataset_min = tf.keras.utils.image_dataset_from_directory(train_dir_min,
                                                                shuffle=True,
                                                                ␣
 ↪batch_size=BATCH_SIZE,
                                                                image_size=IMG_SIZE)
```

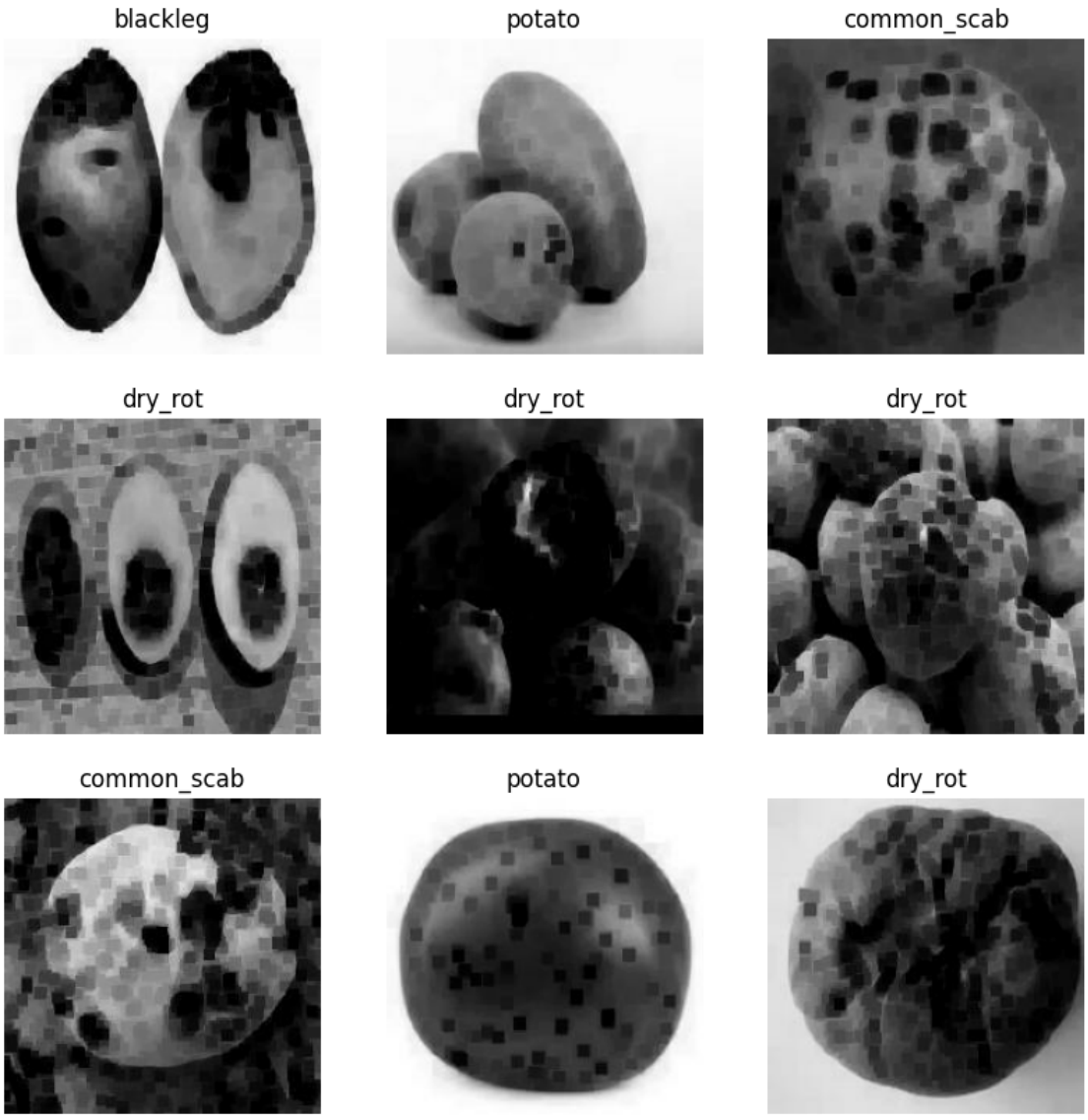
```
Found 209 files belonging to 4 classes.
```

[ ]:
```
validation_dataset_min = tf.keras.utils.
 ↪image_dataset_from_directory(validation_dir_min,
                                                                 shuffle=True,
                                                                ␣
 ↪batch_size=BATCH_SIZE,
                                                                ␣
 ↪image_size=IMG_SIZE)

test_dataset_min = tf.keras.utils.image_dataset_from_directory(test_dir_min,
                                                                 shuffle=True,
                                                                ␣
 ↪batch_size=BATCH_SIZE,
                                                                ␣
 ↪image_size=IMG_SIZE)
```

```
Found 26 files belonging to 4 classes.
Found 27 files belonging to 4 classes.
```

Se visualiza una muestra de las imágenes del conjunto de entrenamiento y sus respectivas etiquetas en una cuadrícula de 3x3.

[ ]:
```
class_names_min = train_dataset_min.class_names

plt.figure(figsize=(10, 10))
for images, labels in train_dataset_min.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names_min[labels[i]])
    plt.axis("off")
```



Al observar las imágenes se puede asumir que el oscurecimiento causado por el filtro Mínimo dificultará enormemente la distinción de las papas enfermas, debido a que este filtro 'neutraliza' sus características distintivas.

[ ]:
```
# Ajusta parametros de memoria automaticamente a la capacidad del computador
AUTOTUNE = tf.data.AUTOTUNE

train_dataset_min = train_dataset_min.prefetch(buffer_size=AUTOTUNE)
validation_dataset_min = validation_dataset_min.prefetch(buffer_size=AUTOTUNE)
test_dataset_min = test_dataset_min.prefetch(buffer_size=AUTOTUNE)
```

[ ]:
```
# Aplica data augmentation al dataset de papas

data_augmentation_min = tf.keras.Sequential([
  tf.keras.layers.RandomFlip('horizontal'),
  tf.keras.layers.RandomRotation(0.1),
  tf.keras.layers.RandomTranslation(height_factor=0.1, width_factor=0.1),
  tf.keras.layers.RandomContrast(factor=0.1),
])
```

[ ]:
```
# Define el modelo que será aplicado para transformar las imágenes en vectores
preprocess_input = tf.keras.applications.resnet_v2.preprocess_input
```

[ ]:
```
# Crea el modelo base a partir del modelo pre-entrenado a partir del modelo␣
 ↪ResNet50V2
IMG_SHAPE = IMG_SIZE + (3,)
base_model_min = tf.keras.applications.ResNet50V2(input_shape=IMG_SHAPE,
                                                  include_top=False,
                                                  weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/resnet50v2_weights_tf_dim_ordering_tf_kernels_notop.h5
94668760/94668760              5s
0us/step
```

En este paso se convierte a las imágenes en bloques de características de tamaño 7x7x2048.

[ ]:
```
image_batch, label_batch = next(iter(train_dataset_min))
feature_batch = base_model_min(image_batch)
print(feature_batch.shape)
```

```
(9, 7, 7, 2048)
```

[ ]:
```
base_model_min.trainable = False
```

[ ]:
```
# Agrega capa de clasificación para transformar las características en vectores␣
 ↪de 2048 elementos

global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(9, 2048)
```

[ ]:
```
# Capa de predicción para transformar las características en una sola␣
 ↪predicción por imagen
num_classes= len(class_names_min)
prediction_layer = tf.keras.layers.Dense(num_classes, activation='softmax')
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(9, 4)
```

[ ]:
```
# Concatena data augmentation, preprocesamiento, el modelo base, y capas que␣
 ↪extraen características.

inputs = tf.keras.Input(shape=(200, 200, 3))
x = data_augmentation_min(inputs)
x = preprocess_input(x)
x = base_model_min(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model_min = tf.keras.Model(inputs, outputs)
```

[ ]:
```
base_learning_rate = 0.0001
model_min.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=False,
              ignore_class=None, reduction='sum_over_batch_size',␣
 ↪name='sparse_categorical_crossentropy'),
              metrics= ['accuracy'])
```

[ ]:
```
initial_epochs = 10

loss0, accuracy0 = model_min.evaluate(validation_dataset_min)
```

```
3/3             14s 3s/step -
accuracy: 0.3921 - loss: 1.3724
```

[ ]:
```
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
initial loss: 1.31
initial accuracy: 0.42
```

[ ]:
```
# guardar el modelo durante el entrenamiento de la red neuronal.

save_checkpoint_5 = tf.keras.callbacks.ModelCheckpoint(
    filepath='./models/model_min_ResNet50V2.keras',␣
 ↪monitor='val_loss',mode='min', save_best_only=True, verbose=1
    )
```

[ ]:
```
history = model_min.fit(train_dataset_min,
                    epochs=initial_epochs,
                    validation_data=validation_dataset_min)
```

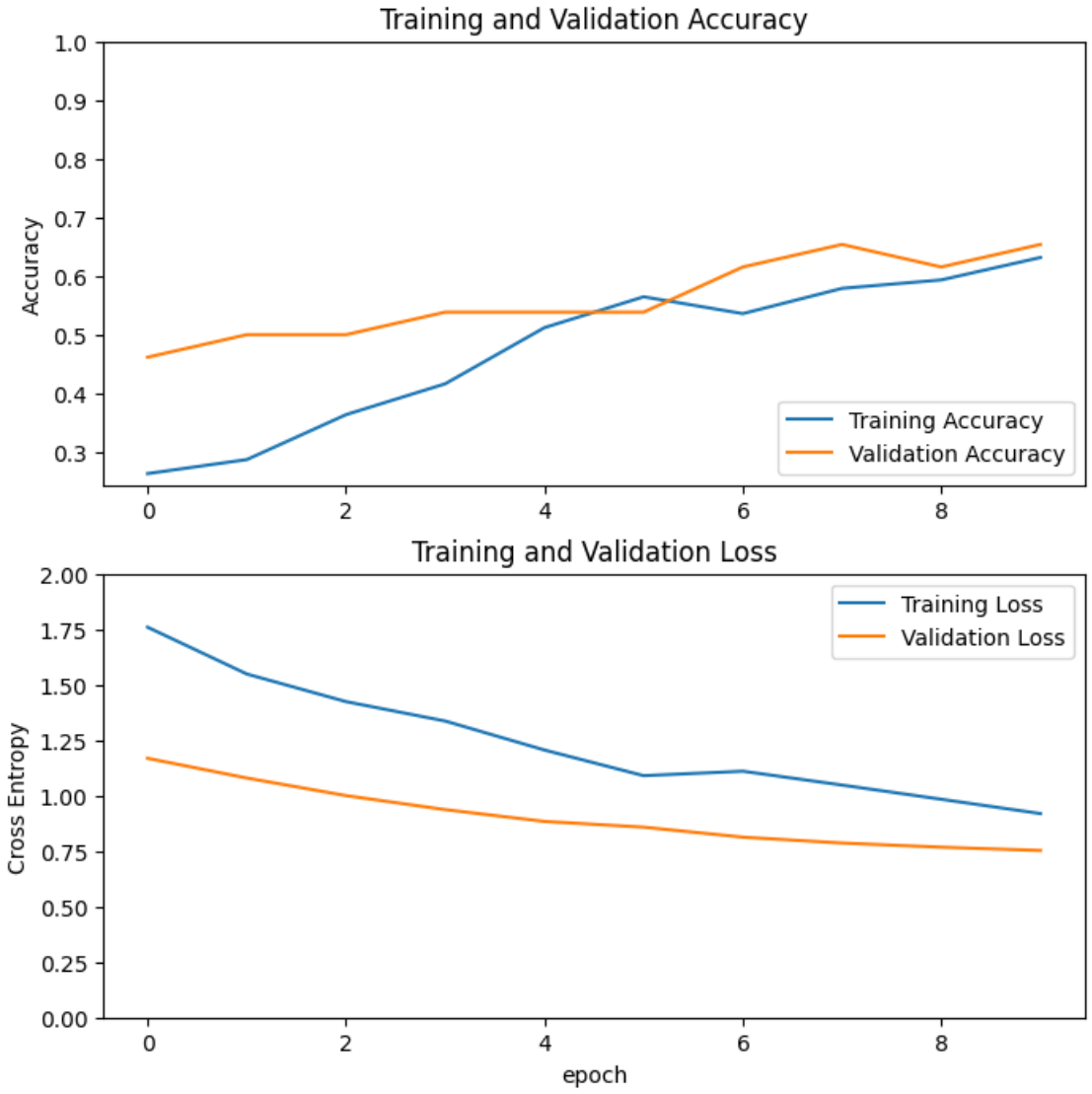
```
Epoch 1/10
24/24             76s 3s/step -
```

```
accuracy: 0.2088 - loss: 1.8333 - val_accuracy: 0.4615 - val_loss: 1.1682
Epoch 2/10
24/24             13s 46ms/step -
accuracy: 0.2511 - loss: 1.5892 - val_accuracy: 0.5000 - val_loss: 1.0794
Epoch 3/10
24/24             1s 43ms/step -
accuracy: 0.3475 - loss: 1.4780 - val_accuracy: 0.5000 - val_loss: 1.0000
Epoch 4/10
24/24             1s 44ms/step -
accuracy: 0.4151 - loss: 1.2999 - val_accuracy: 0.5385 - val_loss: 0.9369
Epoch 5/10
24/24             1s 43ms/step -
accuracy: 0.5342 - loss: 1.1664 - val_accuracy: 0.5385 - val_loss: 0.8837
Epoch 6/10
24/24             1s 43ms/step -
accuracy: 0.5305 - loss: 1.0961 - val_accuracy: 0.5385 - val_loss: 0.8578
Epoch 7/10
24/24             1s 52ms/step -
accuracy: 0.5216 - loss: 1.1357 - val_accuracy: 0.6154 - val_loss: 0.8128
Epoch 8/10
24/24             1s 50ms/step -
accuracy: 0.6073 - loss: 0.9759 - val_accuracy: 0.6538 - val_loss: 0.7863
Epoch 9/10
24/24             1s 53ms/step -
accuracy: 0.6261 - loss: 0.9319 - val_accuracy: 0.6154 - val_loss: 0.7675
Epoch 10/10
24/24             2s 45ms/step -
accuracy: 0.6116 - loss: 0.8827 - val_accuracy: 0.6538 - val_loss: 0.7529
```

[ ]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']
```

[ ]:
```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
```

```
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,2.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



La curva de pérdida del conjunto de entrenamiento presenta algunas fluctuaciones, mientras que la curva de pérdida del conjunto de validación tiene una pendiente suave.

[ ]:
```
# Permite entrenamiento del modelo base
base_model_min.trainable = True
```

[ ]:
```
# Verifica cantidad de capas del modelo base
print("Number of layers in the base model: ", len(base_model_min.layers))

# Fine-tune  desde esta capa
fine_tune_at = 189

# Congela capas anteriores a la capa desde la que se realizará el fine-tuning
for layer in base_model_min.layers[:fine_tune_at]:
  layer.trainable = False
```

```
Number of layers in the base model:  190
```

[ ]:
```
model_min.compile(loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=False,
              ignore_class=None, reduction='sum_over_batch_size',␣
 ↪name='sparse_categorical_crossentropy'),
              optimizer = tf.keras.optimizers.
 ↪RMSprop(learning_rate=base_learning_rate/10),
              metrics = ['accuracy'])
```

[ ]:
```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model_min.fit(train_dataset_min,
                         epochs=total_epochs,
                         initial_epoch=len(history.epoch),
                         validation_data=validation_dataset_min,
                         callbacks=[save_checkpoint_5])
```

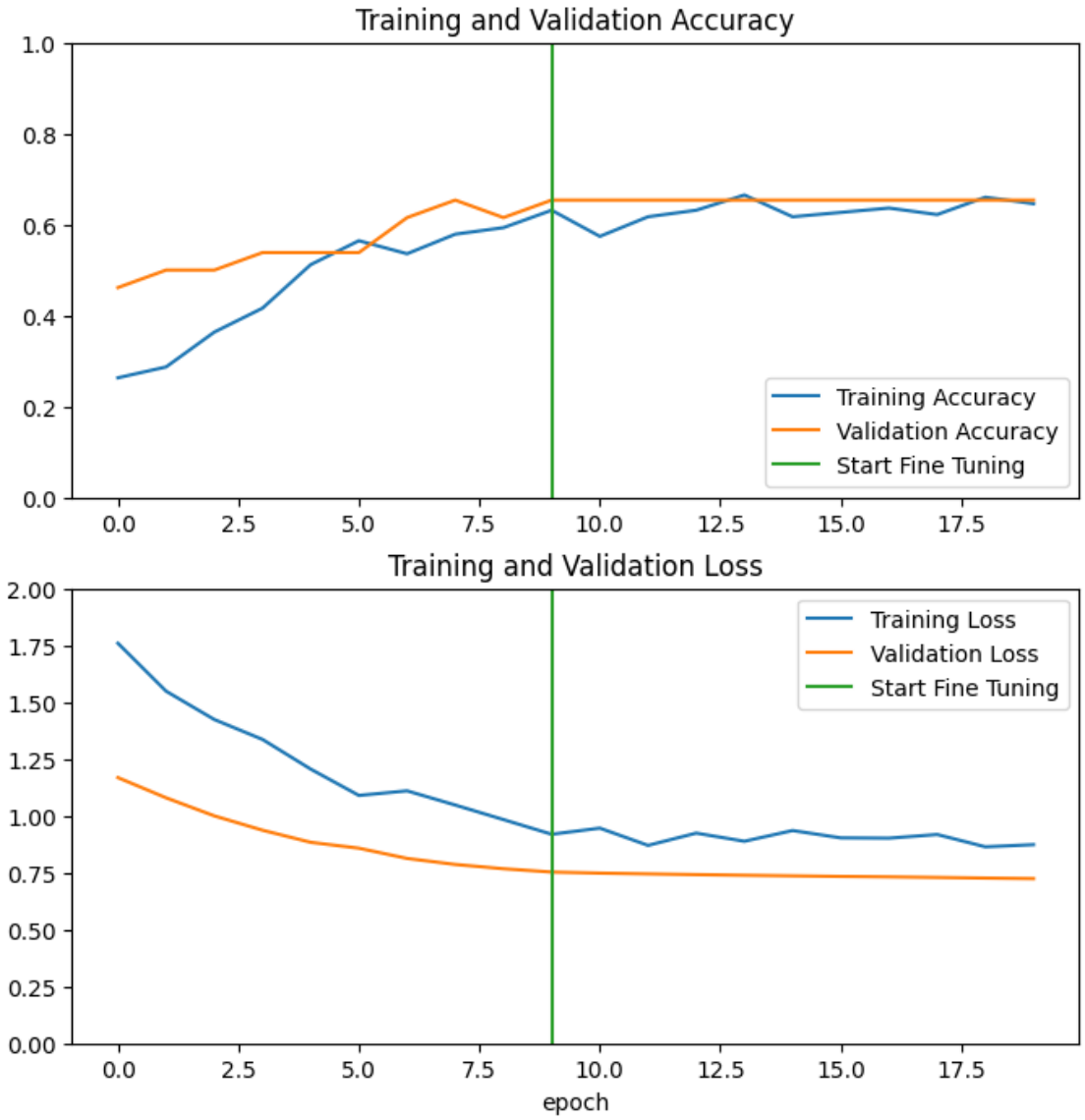
```
Epoch 11/20
24/24             0s 39ms/step -
accuracy: 0.5477 - loss: 0.9322
Epoch 11: val_loss improved from inf to 0.74830, saving model to
./models/model_min_ResNet50V2.keras
24/24             12s 170ms/step -
accuracy: 0.5488 - loss: 0.9327 - val_accuracy: 0.6538 - val_loss: 0.7483
Epoch 12/20
24/24             0s 39ms/step -
accuracy: 0.6548 - loss: 0.8360
Epoch 12: val_loss improved from 0.74830 to 0.74492, saving model to
./models/model_min_ResNet50V2.keras
24/24             2s 77ms/step -
accuracy: 0.6533 - loss: 0.8374 - val_accuracy: 0.6538 - val_loss: 0.7449
Epoch 13/20
24/24             0s 52ms/step -
accuracy: 0.6684 - loss: 0.8591
Epoch 13: val_loss improved from 0.74492 to 0.74152, saving model to
./models/model_min_ResNet50V2.keras
```

```
24/24            3s 92ms/step -
accuracy: 0.6669 - loss: 0.8617 - val_accuracy: 0.6538 - val_loss: 0.7415
Epoch 14/20
24/24            0s 49ms/step -
accuracy: 0.7117 - loss: 0.8176
Epoch 14: val_loss improved from 0.74152 to 0.73877, saving model to
./models/model_min_ResNet50V2.keras
24/24            3s 100ms/step -
accuracy: 0.7099 - loss: 0.8205 - val_accuracy: 0.6538 - val_loss: 0.7388
Epoch 15/20
24/24            0s 57ms/step -
accuracy: 0.6531 - loss: 0.8335
Epoch 15: val_loss improved from 0.73877 to 0.73613, saving model to
./models/model_min_ResNet50V2.keras
24/24            2s 96ms/step -
accuracy: 0.6517 - loss: 0.8376 - val_accuracy: 0.6538 - val_loss: 0.7361
Epoch 16/20
24/24            0s 49ms/step -
accuracy: 0.6636 - loss: 0.8573
Epoch 16: val_loss improved from 0.73613 to 0.73366, saving model to
./models/model_min_ResNet50V2.keras
24/24            2s 89ms/step -
accuracy: 0.6621 - loss: 0.8592 - val_accuracy: 0.6538 - val_loss: 0.7337
Epoch 17/20
24/24            0s 38ms/step -
accuracy: 0.6564 - loss: 0.8587
Epoch 17: val_loss improved from 0.73366 to 0.73177, saving model to
./models/model_min_ResNet50V2.keras
24/24            3s 77ms/step -
accuracy: 0.6556 - loss: 0.8604 - val_accuracy: 0.6538 - val_loss: 0.7318
Epoch 18/20
23/24            0s 50ms/step -
accuracy: 0.6538 - loss: 0.8771
Epoch 18: val_loss improved from 0.73177 to 0.72926, saving model to
./models/model_min_ResNet50V2.keras
24/24            2s 87ms/step -
accuracy: 0.6512 - loss: 0.8804 - val_accuracy: 0.6538 - val_loss: 0.7293
Epoch 19/20
23/24            0s 51ms/step -
accuracy: 0.6697 - loss: 0.8163
Epoch 19: val_loss improved from 0.72926 to 0.72646, saving model to
./models/model_min_ResNet50V2.keras
24/24            2s 90ms/step -
accuracy: 0.6690 - loss: 0.8201 - val_accuracy: 0.6538 - val_loss: 0.7265
Epoch 20/20
23/24            0s 55ms/step -
accuracy: 0.6395 - loss: 0.8631
Epoch 20: val_loss improved from 0.72646 to 0.72408, saving model to
```

```
./models/model_min_ResNet50V2.keras
24/24              3s 106ms/step -
accuracy: 0.6400 - loss: 0.8639 - val_accuracy: 0.6538 - val_loss: 0.7241
```

[ ]:
```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']
```

[ ]:
```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0, 1])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 2.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Las curvas del conjunto de entrenamiento fluctúan, mientras que las del conjunto de validación se mantienen estables durante el proceso de fine-tuning.

[ ]:
```
reconstructed_model_5 = tf.keras.models.load_model('./models/
↪model_min_ResNet50V2.keras')
```

[ ]:
```
score5 = reconstructed_model_5.evaluate(validation_dataset_min, verbose=False)
print('Val loss:', score5[0])
print('Val accuracy:', score5[1])

score5 = reconstructed_model_5.evaluate(test_dataset_min, verbose=False)
print('Test loss:', score5[0])
```

```
print('Test accuracy:', score5[1])
```

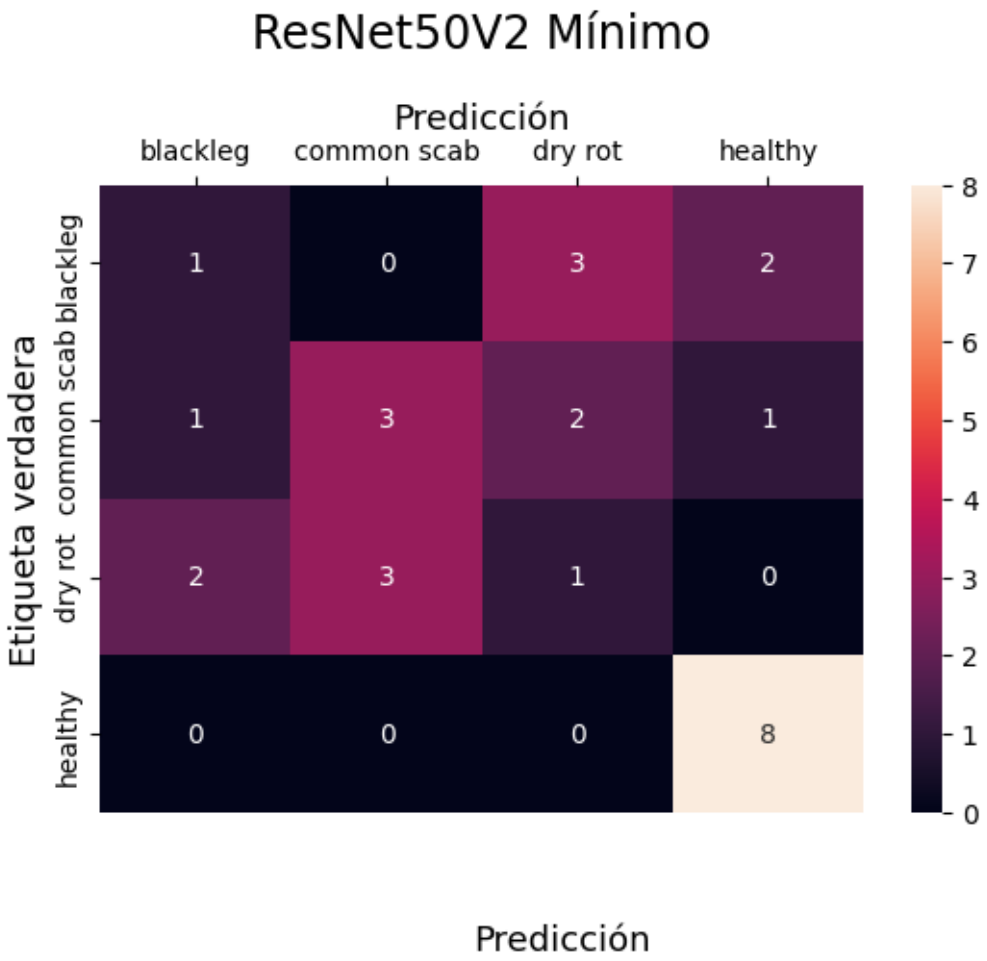
```
Val loss: 0.7240831851959229
Val accuracy: 0.6538461446762085
Test loss: 1.009563684463501
Test accuracy: 0.48148149251937866
```

Los valores de pérdida de los conjuntos de validación y entrenamiento son muy altos, por lo tanto, esto implica que el modelo no aprendió a reconocer correctamente las clases de papas. Además, es sintomático que el valor de accuracy de el set de prueba sea bastante más bajo al del set de validación.

3.8.1 Matriz de confusión y métricas

[ ]:
```
def get_true_labels_and_predictions(model, dataset):
    true_labels = []
    predictions = []
    for images, labels in dataset:
        true_labels.extend(labels.numpy())
        preds = model.predict(images)
        predictions.extend(np.argmax(preds, axis=1))
    return np.array(true_labels), np.array(predictions)

y_test_5, y_pred_5 = get_true_labels_and_predictions(reconstructed_model_5,␣
 ↪test_dataset_min)
cm = confusion_matrix(y_test_5, y_pred_5)
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['blackleg', 'common scab', 'dry rot', 'healthy'],
            yticklabels=['blackleg', 'common scab', 'dry rot', 'healthy'])

plt.ylabel('Etiqueta verdadera', fontsize=13)
plt.title('ResNet50V2 Mínimo', fontsize=17, pad=20)
plt.gca().xaxis.set_label_position('top')
plt.xlabel('Predicción', fontsize=13)
plt.gca().xaxis.tick_top()

plt.gca().figure.subplots_adjust(bottom=0.2)
plt.gca().figure.text(0.5, 0.05, 'Predicción', ha='center', fontsize=13)
plt.show()
```

```
1/1             2s 2s/step
1/1             0s 77ms/step
1/1             0s 75ms/step
```



Una papa infectada por blackleg fue clasificada correctamente, siendo tres muestras etiquetadas como papas infectadas por dry rot, y dos clasificadas como papas sanas.

Tres papas con common scab fueron clasificadas correctamente, siendo una etiquetada como papa con blackleg, dos como infectadas por dry rot y una como papa sana.

Solamente una papa con dry rot fue etiquetada correctamente, siendo dos clasificadas como papas con blackleg, y tres etiquetadas como papas infectadas por common scab.

Todas las papas sanas fueron clasificadas correctamente.

[ ]:
```
def c_report(y_true, y_pred, class_names):
    report = classification_report(y_true, y_pred, target_names=class_names,␣
 ↪zero_division=1)
    return report

report_5 = c_report(y_test_5, y_pred_5, class_names_min)
```

```
print(f"Classification Report for Model 5:\n{report_5}")
```

```
Classification Report for Model 5:
              precision    recall  f1-score   support

    blackleg       0.25      0.17      0.20         6
 common_scab       0.50      0.43      0.46         7
     dry_rot       0.17      0.17      0.17         6
      potato       0.73      1.00      0.84         8

    accuracy                           0.48        27
   macro avg       0.41      0.44      0.42        27
weighted avg       0.44      0.48      0.45        27
```

Como se ha observado en los modelos anteriores, en este caso los mejores resultados corresponden a la clase potato. Los peores resultados corresponden a las clases dry rot y blackleg.

3.8.2 Determinación del tiempo de ejecución del proceso de fine-tuning del modelo ResNet50V2 para el 50% y el 100% de las imágenes del dataset

El tiempo de procesamiento del 50% y 100% de las imágenes del dataset considera el uso de una tarjeta Nvidia T4 para acelerarlo. Sólo se mide cinco veces debido a que la GPU es un recurso costoso de utilizar.

[ ]:
```
ex_time = []
half_time = []

j = 1

while j < 6:
    start = time.perf_counter()


    fine_tune_epochs = 10
    total_epochs =  initial_epochs + fine_tune_epochs

    history_fine = model_min.fit(train_dataset_min,
                         epochs=total_epochs,
                         initial_epoch=len(history.epoch),
                         validation_data=validation_dataset_min)

    end = time.perf_counter()
    total = end - start
    print(f"Tiempo transcurrido: {(total)} s")
    ex_time.append(total)
    half_t = (total/2)
    half_time.append(half_t)
```

```
    j +=1
```

[ ]:
```
print("Tiempo mínimo ", min(ex_time), 's')
print("Tiempo máximo: ", max(ex_time), 's')
print("Tiempo promedio: ", sum(ex_time)/len(ex_time),'s')
print("Desviación estándar del tiempo: ", np.std(ex_time), 's')
```

```
Tiempo mínimo  11.736598716999993 s
Tiempo máximo:  13.491030301000023 s
Tiempo promedio:  12.46160138560001 s
Desviación estándar del tiempo:  0.6506057967863514 s
```

[ ]:
```
print("Tiempo mínimo mitad ejecución ", min(half_time), 's')
print("Tiempo máximo mitad ejecución: ", max(half_time), 's')
print("Tiempo promedio mitad ejecución: ", sum(half_time)/len(half_time), 's')
print("Desviación estándar del tiempo mitad ejecución: ", np.std(half_time),␣
 ↪'s')
```

```
Tiempo mínimo mitad ejecución  5.868299358499996 s
Tiempo máximo mitad ejecución:  6.745515150500012 s
Tiempo promedio mitad ejecución:  6.230800692800005 s
Desviación estándar del tiempo mitad ejecución:  0.3253028983931757 s
```

Luego de realizar fine-tuning utilizando el modelo ResNet50V2 como base, se comprueba que los resultados obtenidos usando el dataset sin filtrar tienen un error que no permite confiar en los resultados obtenidos.

Tomando en cuenta el alto nivel de error observado, los modelos distinguen entre las papas sanas y las papas enfermas, pero presentan problemas a la hora de determinar correctamente el tipo de enfermedad.

También se comprueba que los resultados de los datasets filtrado mediante desenfoque Gaussiano, detección de bordes Sobel, filtros Máximo y Mínimo son peores que los del dataset sin filtrar, dado que se trata de imágenes en escala de grises desenfocadas en el caso del filtro Gaussiano, imágenes que resaltan los bordes en el caso del filtro Sobel, imágenes aclaradas en el caso del filtro Máximo e imágenes oscurecidas en el caso del filtro Mínimo, divergiendo de los colores de las imágenes utilizadas para pre-entrenar el modelo ResNet50V2. Se concluye que tanto la forma como los colores de las lesiones son cruciales para diferenciar los tres tipos de papas enfermas, características que se ven alteradas al aplicar los filtros Máximo, Mínimo y Gaussiano. En el caso del filtro Sobel sólo se conservan los bordes de las papas, perdiendo información vital para su correcta clasificación.

Además, a pesar de aplicar data augmentation, el dataset sólo contiene 262 imágenes en total, lo cual no es un tamaño de muestra adecuado para un modelo de deep learning. Se sugiere aumentar sustancialmente la cantidad de muestras para obtener mejores resultados a futuro.

3.9 Clasificador AdaBoost para dataset sin filtrar

El clasificador AdaBoost consiste en crear varios predictores sencillos en secuencia, de tal manera que el segundo predictor ajuste bien lo que el primero no ajustó, que el tercer predictor ajuste lo que el segundo no pudo ajustar y así sucesivamente. En este caso el predictor débil es un árbol de clasificación con profundidad de nodo igual a uno.

[ ]:
```
Categoría=['blackleg', 'common_scab', 'dry_rot', 'potato'] # etiquetas papas␣
↪enfermas y sanas
```

La determinación del tiempo de procesamiento del clasificador AdaBoost fue realizada 5 veces, considerando desde la transformación de las imágenes a arrays hasta completar la predicción de las cuatro clases de papas. El tiempo del procesamiento del 50% de las imágenes será calculado dividiendo el tiempo total por dos. Las imágenes del dataset fueron transformadas en arrays 1D mediante embedding directo antes de realizar su clasificación.

[ ]:
```
ex_time_nf = []
half_time_nf = []

j = 1

while j < 6:
    start = time.perf_counter()

    flat_data_arr=[]
    target_arr = []
    datadir ='/content/drive/MyDrive/potato/data'

    for i in Categoría:
        print(f'Cargando categoría: {i}')
        path=os.path.join(datadir,i)
        for img in os.listdir(path):
            img_array=imread(os.path.join(path,img)) # lee cada imagen del␣
 ↪dataset
            img_resized=resize(img_array,(200,200,3))   # asegura que las␣
 ↪imágenes tengan las mismas dimensiones y canales
            flat_data_arr.append(img_resized.flatten()) # transforma array en␣
 ↪array 1D
            target_arr.append(Categoría.index(i)) # agrega los índices de cada␣
 ↪etiqueta
    flat_data=np.array(flat_data_arr) # crea array de Numpy para imágenes
    target=np.array(target_arr) # crea array de Numpy para etiquetas
    df_papa=pd.DataFrame(flat_data) # genera dataframe a partir de los arrays␣
 ↪de las imágenes
    df_papa['Categoría']=target # agrega columna con las etiquetas

    x=df_papa.iloc[:,:-1]
    y=df_papa.iloc[:,-1]
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
 ↪20,random_state=42) # divide los datos en train y test asegurando que los␣
 ↪resultados sean reproducibles
```

```
    modeloAdab = AdaBoostClassifier(estimator =␣
↪DecisionTreeClassifier(max_depth=1), n_estimators=50,random_state=123).
↪fit(x_train,y_train)
    y_preds = modeloAdab.predict(x_test)

    end = time.perf_counter()
    total = end - start  # calcula tiempo total
    print(f"Tiempo transcurrido: {(total)} s")
    ex_time_nf.append(total)
    half_t = (total/2)  # calcula valor para aproximar el tiempo que toma␣
↪procesar el 50% de las imágenes
    half_time_nf.append(half_t)
    j +=1
```

```
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 211.66433405099997 s
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 216.7520622359998 s
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 265.1938772420008 s
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 227.0675679799997 s
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 192.48713796100037 s
```

3.9.1 Determinación del tiempo de ejecución para el 50% y 100% de las imágenes clasificadas

[ ]:
```
print("Tiempo mínimo ", min(ex_time_nf), 's')
print("Tiempo máximo: ", max(ex_time_nf), 's')
print("Tiempo promedio: ", sum(ex_time_nf)/len(ex_time_nf), 's')
print("Desviación estándar del tiempo: ", np.std(ex_time_nf), 's')
```

```
Tiempo mínimo  192.48713796100037 s
Tiempo máximo:  265.1938772420008 s
Tiempo promedio:  222.63299589400012 s
Desviación estándar del tiempo:  24.06143688368178 s
```

[ ]:
```
print("Tiempo mínimo mitad ejecución ", min(half_time_nf), 's')
print("Tiempo máximo mitad ejecución: ", max(half_time_nf), 's')
print("Tiempo promedio mitad ejecución: ", sum(half_time_nf)/len(half_time_nf),␣
↪'s')
print("Desviación estándar del tiempo mitad ejecución: ", np.std(half_time_nf),␣
↪'s')
```

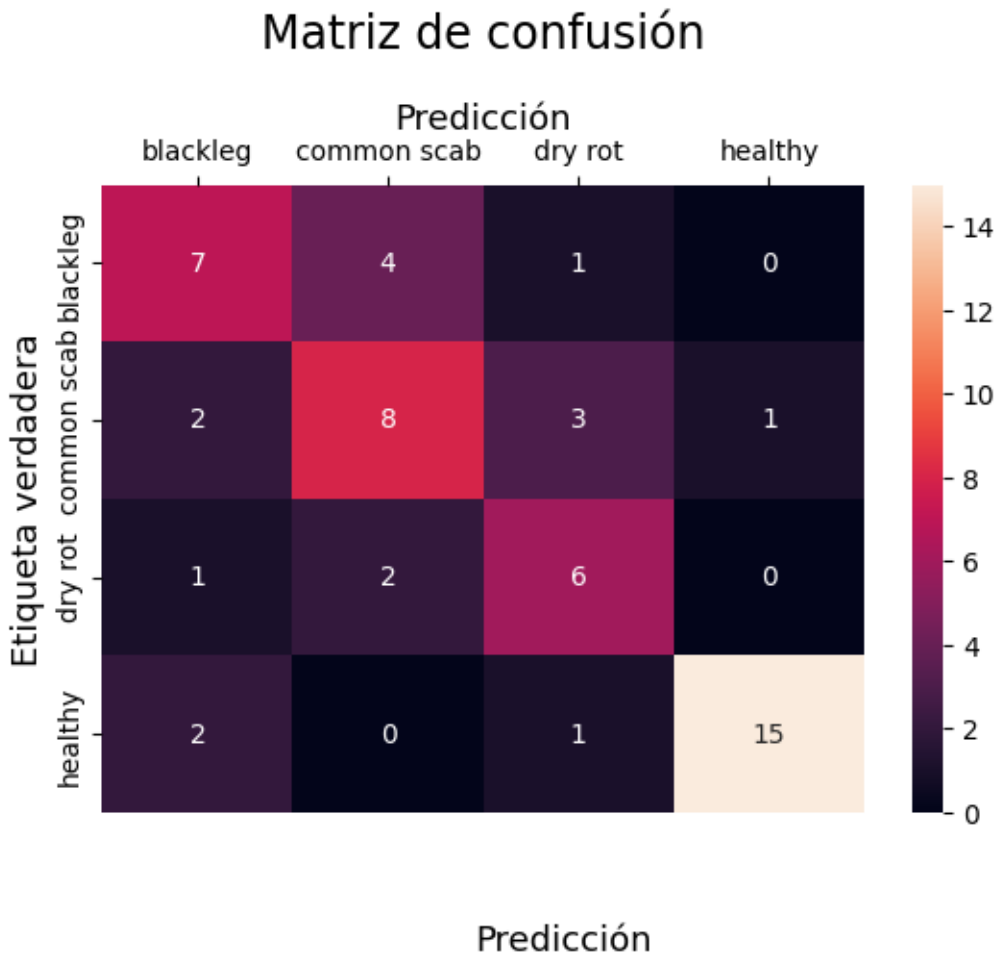
```
Tiempo mínimo mitad ejecución  96.24356898050019 s
Tiempo máximo mitad ejecución:  132.5969386210004 s
Tiempo promedio mitad ejecución:  111.31649794700006 s
Desviación estándar del tiempo mitad ejecución:  12.03071844184089 s
```

3.9.2 Matriz de confusión y métricas de clasificación correspondientes a las imágenes sin filtrar

[ ]:
```
y_preds
```

[ ]:
```
array([3, 0, 3, 1, 1, 2, 2, 3, 0, 1, 1, 3, 0, 3, 3, 3, 2, 0, 3, 1, 2, 0,
       1, 0, 1, 1, 2, 1, 0, 1, 0, 3, 2, 2, 0, 3, 1, 3, 3, 1, 3, 2, 1, 3,
       0, 2, 3, 0, 0, 2, 3, 2, 1])
```

[ ]:
```
# Matriz de confusión para SVM aplicado a imágenes no filtradas
cm = confusion_matrix(y_test, y_preds)
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['blackleg', 'common scab', 'dry rot', 'healthy'],
            yticklabels=['blackleg', 'common scab', 'dry rot', 'healthy'])

plt.ylabel('Etiqueta verdadera', fontsize=13)
plt.title('Matriz de confusión', fontsize=17, pad=20)
plt.gca().xaxis.set_label_position('top')
plt.xlabel('Predicción', fontsize=13)
plt.gca().xaxis.tick_top()

plt.gca().figure.subplots_adjust(bottom=0.2)
plt.gca().figure.text(0.5, 0.05, 'Predicción', ha='center', fontsize=13)
plt.show()
```



Las papas infectadas por blackleg fueron confundidas en cuatro casos por papas infectadas por common scab y en un caso como infectada por dry rot. En el caso de las papas infectadas por common scab, dos fueron clasificadas erróneamente como papas infectadas por blackleg, tres fueron clasificadas como papas infectadas por dry rot, y una fue clasificada como una papa sana. En el caso de las papas infectadas por dry rot, dos fueron clasificadas como papas infectadas por blackleg, y dos como infectadas por common scab. En el caso de las papas sanas, casi todas fueron clasificadas correctamente, siendo dos papas clasificadas comos papas infectadas con blackleg y una papa como infectada por dry rot.

[ ]:
```
print('Accuracy  :', accuracy_score(y_test, y_preds)*100 ,"%")
print('Precision :' , precision_score(y_test, y_preds, average='weighted')*100␣
↪,"%")
print('Recall    :', recall_score(y_test, y_preds, average='weighted')*100 ,"%")
print('F1 Score  :',f1_score(y_test, y_preds, average='weighted')*100 ,"%")
```

```
Accuracy  : 67.9245283018868 %
Precision : 69.40394511149228 %
Recall    : 67.9245283018868 %
```

```
F1 Score  : 68.45726970033296 %
```

3.10 Clasificador AdaBoost para dataset filtro Gaussiano

La determinación del tiempo de procesamiento del clasificador AdaBoost fue realizada 5 veces, considerando desde la transformación de las imágenes a arrays hasta completar la predicción de las cuatro clases de papas. El tiempo del procesamiento del 50% de las imágenes será calculado dividiendo el tiempo total por dos. Las imágenes del dataset fueron transformadas en arrays 1D mediante embedding directo antes de realizar su clasificación.

[ ]:
```
ex_time = []
half_time = []

j = 1

while j < 6:
    start = time.perf_counter()

    flat_data_arr_g=[]
    target_arr_g=[]
    datadir_g='/content/drive/MyDrive/potato/data_g'

    for i in Categoría:
        print(f'Cargando categoría: {i}')
        path=os.path.join(datadir_g,i)
        for img in os.listdir(path):
            img_array=imread(os.path.join(path,img))
            img_resized=resize(img_array,(200,200,3))
            flat_data_arr_g.append(img_resized.flatten())
            target_arr_g.append(Categoría.index(i))
    flat_data_g=np.array(flat_data_arr_g)
    target_g=np.array(target_arr_g)
    df_papa_g=pd.DataFrame(flat_data_g)
    df_papa_g['Categoría']=target_g
    df_papa_g

    x_g=df_papa_g.iloc[:,:-1]
    y_g=df_papa_g.iloc[:,-1]
    x_g_train,x_g_test,y_g_train,y_g_test=train_test_split(x_g,y_g,test_size=0.
↪20,random_state=42)

    modeloAdab = AdaBoostClassifier(estimator =␣
↪DecisionTreeClassifier(max_depth=1), n_estimators=50,random_state=123).
↪fit(x_g_train,y_g_train)
    y_g_preds = modeloAdab.predict(x_g_test)


    end = time.perf_counter()
```

```
    total = end - start
    print(f"Tiempo transcurrido: {(total)} s")
    ex_time.append(total)
    half_t = (total/2)
    half_time.append(half_t)
    j +=1
```

```
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 201.40067848499984 s
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 172.52486111599956 s
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 173.69879389700054 s
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 172.87939980600004 s
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 171.96284872700016 s
```

3.10.1 Determinación del tiempo de ejecución para el 50% y 100% de las imágenes clasificadas

[ ]:
```
print("Tiempo mínimo ", min(ex_time), 's')
print("Tiempo máximo: ", max(ex_time), 's')
print("Tiempo promedio: ", sum(ex_time)/len(ex_time),'s')
print("Desviación estándar del tiempo: ", np.std(ex_time), 's')
```

```
Tiempo mínimo  171.96284872700016 s
Tiempo máximo:  201.40067848499984 s
Tiempo promedio:  178.49331640620002 s
Desviación estándar del tiempo:  11.467521177261052 s
```

[ ]:
```
print("Tiempo mínimo mitad ejecución ", min(half_time), 's')
print("Tiempo máximo mitad ejecución: ", max(half_time), 's')
```

```
print("Tiempo promedio mitad ejecución: ", sum(half_time)/len(half_time), 's')
print("Desviación estándar del tiempo mitad ejecución: ", np.std(half_time),␣
 ↪'s')
```

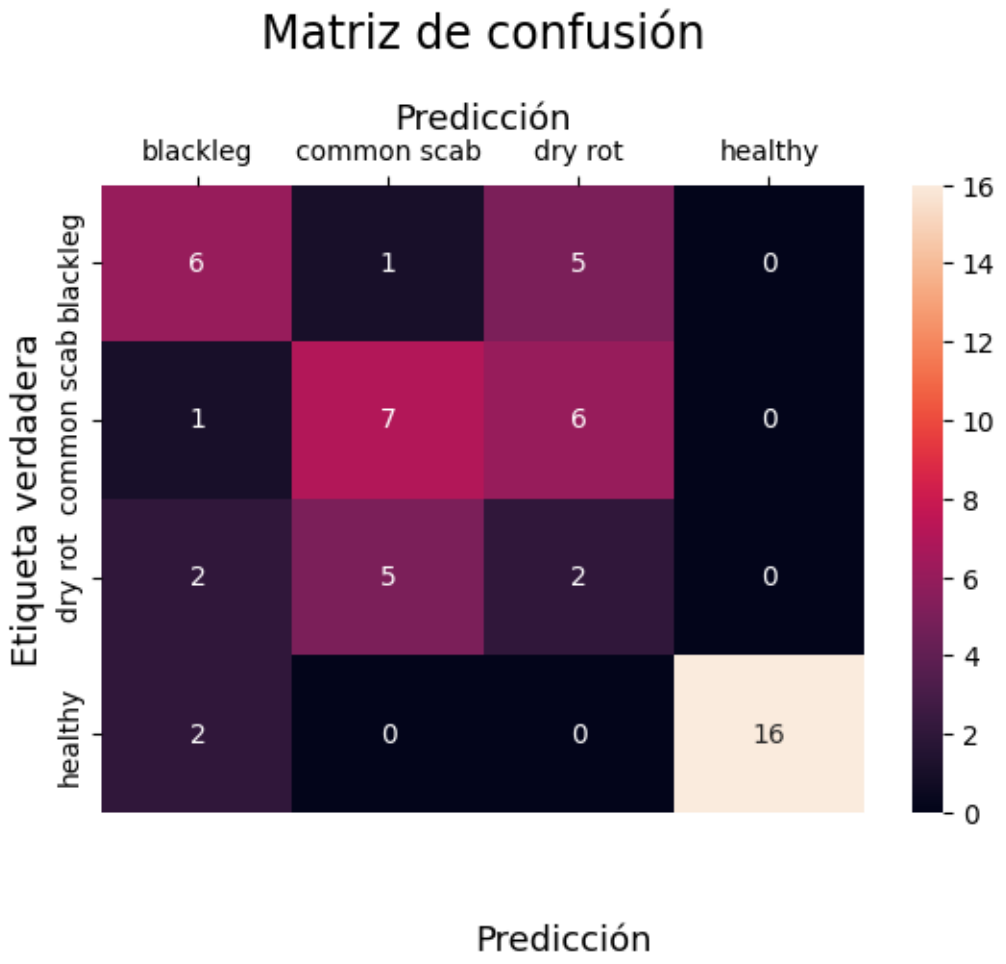
```
Tiempo mínimo mitad ejecución  85.98142436350008 s
Tiempo máximo mitad ejecución:  100.70033924249992 s
Tiempo promedio mitad ejecución:  89.24665820310001 s
Desviación estándar del tiempo mitad ejecución:  5.733760588630526 s
```

3.10.2 Matriz de confusión y métricas de clasificación correspondientes a las imágenes pre-procesadas con filtro Gaussiano

[ ]:
```
y_g_preds
```

[ ]:
```
array([3, 3, 3, 2, 0, 2, 1, 3, 2, 2, 0, 3, 2, 3, 0, 3, 1, 0, 3, 2, 1, 0,
       1, 0, 1, 0, 1, 2, 2, 2, 0, 3, 3, 1, 0, 3, 1, 3, 3, 0, 3, 2, 2, 2,
       2, 1, 3, 0, 1, 1, 3, 1, 1])
```

[ ]:
```
cm = confusion_matrix(y_g_test, y_g_preds)
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['blackleg', 'common scab', 'dry rot', 'healthy'],
            yticklabels=['blackleg', 'common scab', 'dry rot', 'healthy'])

plt.ylabel('Etiqueta verdadera', fontsize=13)
plt.title('Matriz de confusión', fontsize=17, pad=20)
plt.gca().xaxis.set_label_position('top')
plt.xlabel('Predicción', fontsize=13)
plt.gca().xaxis.tick_top()

plt.gca().figure.subplots_adjust(bottom=0.2)
plt.gca().figure.text(0.5, 0.05, 'Predicción', ha='center', fontsize=13)
plt.show()
```



Las papas infectadas por blackleg fueron confundidas en un caso como papa infectada por common scab, y en cinco casos como infectadas por dry rot. En el caso de las papas infectadas por common scab, una fue clasificada como papa infectada por blackleg, y seis fueron clasificadas como papas infectadas por dry rot. En el caso de las papas infectadas por dry rot, dos fueron clasificadas como papas infectadas por blackleg, y cinco como infectadas por common scab. En el caso de las papas sanas, casi todas fueron clasificadas correctamente, siendo dos papas clasificadas como papas infectadas con blackleg.

[ ]:
```
print('Accuracy  :', accuracy_score(y_g_test, y_g_preds)*100 ,"%")
print('Precision :' , precision_score(y_g_test, y_g_preds,␣
 ↪average='weighted')*100 ,"%")
print('Recall    :', recall_score(y_g_test, y_g_preds, average='weighted')*100␣
 ↪,"%")
print('F1 Score  :',f1_score(y_g_test, y_g_preds, average='weighted')*100 ,"%")
```

```
Accuracy  : 58.490566037735846 %
Precision : 63.14817258213484 %
Recall    : 58.490566037735846 %
```

```
F1 Score  : 60.56163949326137 %
```

3.11 Clasificador AdaBoost para dataset filtro Sobel

La determinación del tiempo de procesamiento del clasificador AdaBoost fue realizada 5 veces, considerando desde la transformación de las imágenes a arrays hasta completar la predicción de las cuatro clases de papas. El tiempo del procesamiento del 50% de las imágenes será calculado dividiendo el tiempo total por dos. Las imágenes del dataset fueron transformadas en arrays 1D mediante embedding directo antes de realizar su clasificación.

[ ]:
```
ex_time_s = []
half_time_s = []

j = 1

while j < 6:
    start = time.perf_counter()

    flat_data_arr_s=[]
    target_arr_s=[]
    datadir_s='/content/drive/MyDrive/potato/data_s'

    for i in Categoría:
        print(f'Cargando categoría: {i}')
        path=os.path.join(datadir_s,i)
        for img in os.listdir(path):
            img_array=imread(os.path.join(path,img))
            img_resized=resize(img_array,(200,200,3))
            flat_data_arr_s.append(img_resized.flatten())
            target_arr_s.append(Categoría.index(i))
    flat_data_s=np.array(flat_data_arr_s)
    target_s=np.array(target_arr_s)
    df_papa_s=pd.DataFrame(flat_data_s)
    df_papa_s['Categoría']=target_s
    df_papa_s

    x_s=df_papa_s.iloc[:,:-1]
    y_s=df_papa_s.iloc[:,-1]
    x_s_train,x_s_test,y_s_train,y_s_test=train_test_split(x_s,y_s,test_size=0.
↪20,random_state=42)

    modeloAdab = AdaBoostClassifier(estimator =␣
↪DecisionTreeClassifier(max_depth=1), n_estimators= 50,random_state=123).
↪fit(x_s_train,y_s_train)
    y_s_preds = modeloAdab.predict(x_s_test)

    end = time.perf_counter()
    total = end - start
```

```
    print(f"Tiempo transcurrido: {(total)} s")
    ex_time_s.append(total)
    half_t = (total/2)
    half_time_s.append(half_t)
    j +=1
```

```
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 223.22405721400082 s
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 149.05102069199893 s
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 149.14524732300015 s
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 148.84203224700104 s
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 151.43890395699964 s
```

3.11.1 Determinación del tiempo de ejecución para el 50% y 100% de las imágenes clasificadas

[ ]:
```
print("Tiempo mínimo ", min(ex_time_s), 's')
print("Tiempo máximo: ", max(ex_time_s), 's')
print("Tiempo promedio: ", sum(ex_time_nf)/len(ex_time_s), 's')
print("Desviación estándar del tiempo: ", np.std(ex_time_s), 's')
```

```
Tiempo mínimo  148.84203224700104 s
Tiempo máximo:  223.22405721400082 s
Tiempo promedio:  222.63299589400012 s
Desviación estándar del tiempo:  29.457056453226528 s
```

[ ]:
```
print("Tiempo mínimo mitad ejecución ", min(half_time_s), 's')
print("Tiempo máximo mitad ejecución: ", max(half_time_s), 's')
```

```
print("Tiempo promedio mitad ejecución: ", sum(half_time_s)/len(half_time_s),␣
 ↪'s')
print("Desviación estándar del tiempo mitad ejecución: ", np.std(half_time_s),␣
 ↪'s')
```

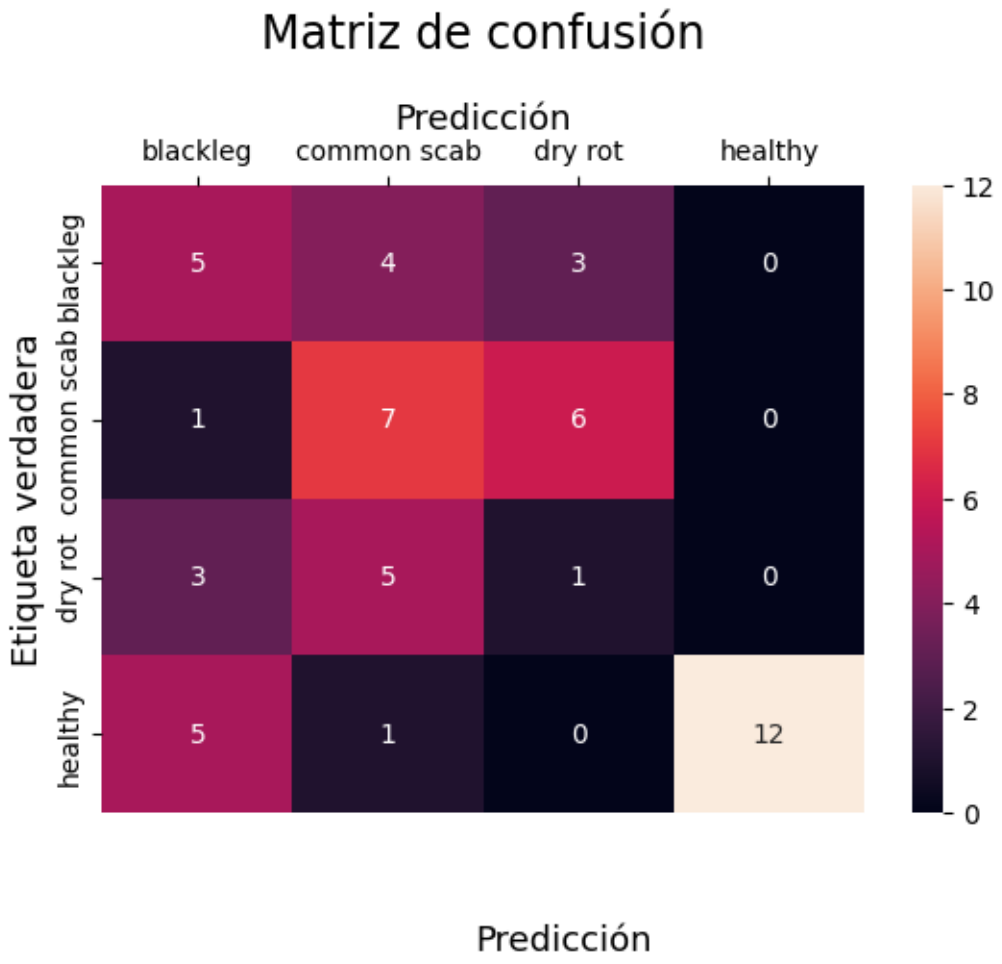
```
Tiempo mínimo mitad ejecución  74.42101612350052 s
Tiempo máximo mitad ejecución:  111.61202860700041 s
Tiempo promedio mitad ejecución:  82.17012614330005 s
Desviación estándar del tiempo mitad ejecución:  14.728528226613264 s
```

3.11.2 Matriz de confusión y métricas de clasificación correspondientes a las imágenes pre-procesadas con filtro Sobel

[ ]:
```
y_s_preds
```

[ ]:
```
array([0, 3, 3, 1, 1, 1, 0, 3, 0, 1, 1, 1, 1, 0, 3, 3, 1, 1, 0, 0, 0, 1,
       0, 0, 2, 1, 2, 1, 1, 2, 0, 3, 3, 1, 2, 3, 2, 0, 3, 0, 3, 1, 2, 2,
       2, 0, 3, 0, 1, 2, 3, 1, 2])
```

[ ]:
```
cm = confusion_matrix(y_s_test, y_s_preds)
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['blackleg', 'common scab', 'dry rot', 'healthy'],
            yticklabels=['blackleg', 'common scab', 'dry rot', 'healthy'])

plt.ylabel('Etiqueta verdadera', fontsize=13)
plt.title('Matriz de confusión', fontsize=17, pad=20)
plt.gca().xaxis.set_label_position('top')
plt.xlabel('Predicción', fontsize=13)
plt.gca().xaxis.tick_top()

plt.gca().figure.subplots_adjust(bottom=0.2)
plt.gca().figure.text(0.5, 0.05, 'Predicción', ha='center', fontsize=13)
plt.show()
```



Las papas infectadas por blackleg fueron confundidas en cuatro casos como papas infectadas por common scab, y en tres casos como infectadas por dry rot. En el caso de las papas infectadas por common scab, una fue clasificada como papa infectada por blackleg, y seis fueron clasificadas como papas infectadas por dry rot. En el caso de las papas infectadas por dry rot, tres fueron clasificadas como papas infectadas por blackleg, y cinco como infectadas por common scab. En el caso de las papas sanas, doce fueron clasificadas correctamente, siendo cinco papas clasificadas como papas infectadas con blackleg, y una papa fue etiquetada como papa infectada por common scab.

[ ]:
```
print('Accuracy  :', accuracy_score(y_s_test, y_s_preds)*100 ,"%")
print('Precision :' , precision_score(y_s_test, y_s_preds,␣
  ↪average='weighted')*100 ,"%")
print('Recall    :', recall_score(y_s_test, y_s_preds, average='weighted')*100␣
  ↪,"%")
print('F1 Score  :',f1_score(y_s_test, y_s_preds, average='weighted')*100 ,"%")
```

```
Accuracy  : 47.16981132075472 %
Precision : 54.62343427937212 %
Recall    : 47.16981132075472 %
```

```
F1 Score  : 49.59496921056328 %
```

3.12 Clasificador AdaBoost para dataset filtro Máximo

La determinación del tiempo de procesamiento del clasificador AdaBoost fue realizada 5 veces, considerando desde la transformación de las imágenes a arrays hasta completar la predicción de las cuatro clases de papas. El tiempo del procesamiento del 50% de las imágenes será calculado dividiendo el tiempo total por dos. Las imágenes del dataset fueron transformadas en arrays 1D mediante embedding directo antes de realizar su clasificación.

[ ]:
```
ex_time_max = []
half_time_max = []

j = 1

while j < 6:
    start = time.perf_counter()

    flat_data_arr_max=[]
    target_arr_max=[]
    datadir_max='/content/drive/MyDrive/potato/data_max'

    for i in Categoría:
        print(f'Cargando categoría: {i}')
        path=os.path.join(datadir_max,i)
        for img in os.listdir(path):
            img_array=imread(os.path.join(path,img))
            img_resized=resize(img_array,(200,200,3))
            flat_data_arr_max.append(img_resized.flatten())
            target_arr_max.append(Categoría.index(i))
    flat_data_max=np.array(flat_data_arr_max)
    target_max=np.array(target_arr_max)
    df_papa_max=pd.DataFrame(flat_data_max)
    df_papa_max['Categoría']=target_max
    df_papa_max

    x_max=df_papa_max.iloc[:,:-1]
    y_max=df_papa_max.iloc[:,-1]
  ␣
↪x_max_train,x_max_test,y_max_train,y_max_test=train_test_split(x_max,y_max,test_size=0.
↪20,random_state=42)

   modeloAdab = AdaBoostClassifier(estimator =␣
↪DecisionTreeClassifier(max_depth=1), n_estimators=50,random_state=123).
↪fit(x_max_train,y_max_train)
   y_max_preds = modeloAdab.predict(x_max_test)
```

```
    end = time.perf_counter()
    total = end - start
    print(f"Tiempo transcurrido: {(total)} s")
    ex_time_max.append(total)
    half_t = (total/2)
    half_time_max.append(half_t)
    j +=1
```

```
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 230.5973333810016 s
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 155.82422517199848 s
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 157.34371942199868 s
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 156.17993759099954 s
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 154.2939979950006 s
```

3.12.1 Determinación del tiempo de ejecución para el 50% y 100% de las imágenes clasificadas

[ ]:
```
print("Tiempo mínimo ", min(ex_time_max), 's')
print("Tiempo máximo: ", max(ex_time_max), 's')
print("Tiempo promedio: ", sum(ex_time_max)/len(ex_time_max), 's')
print("Desviación estándar del tiempo: ", np.std(ex_time_max), 's')
```

```
Tiempo mínimo  154.2939979950006 s
Tiempo máximo:  230.5973333810016 s
Tiempo promedio:  170.8478427121998 s
Desviación estándar del tiempo:  29.890631577730524 s
```

[ ]:
```
print("Tiempo mínimo mitad ejecución ", min(half_time_max), 's')
print("Tiempo máximo mitad ejecución: ", max(half_time_max))
print("Tiempo promedio mitad ejecución: ", sum(half_time_max)/
 ↪len(half_time_max), 's')
print("Desviación estándar del tiempo mitad ejecución: ", np.
 ↪std(half_time_max), 's')
```

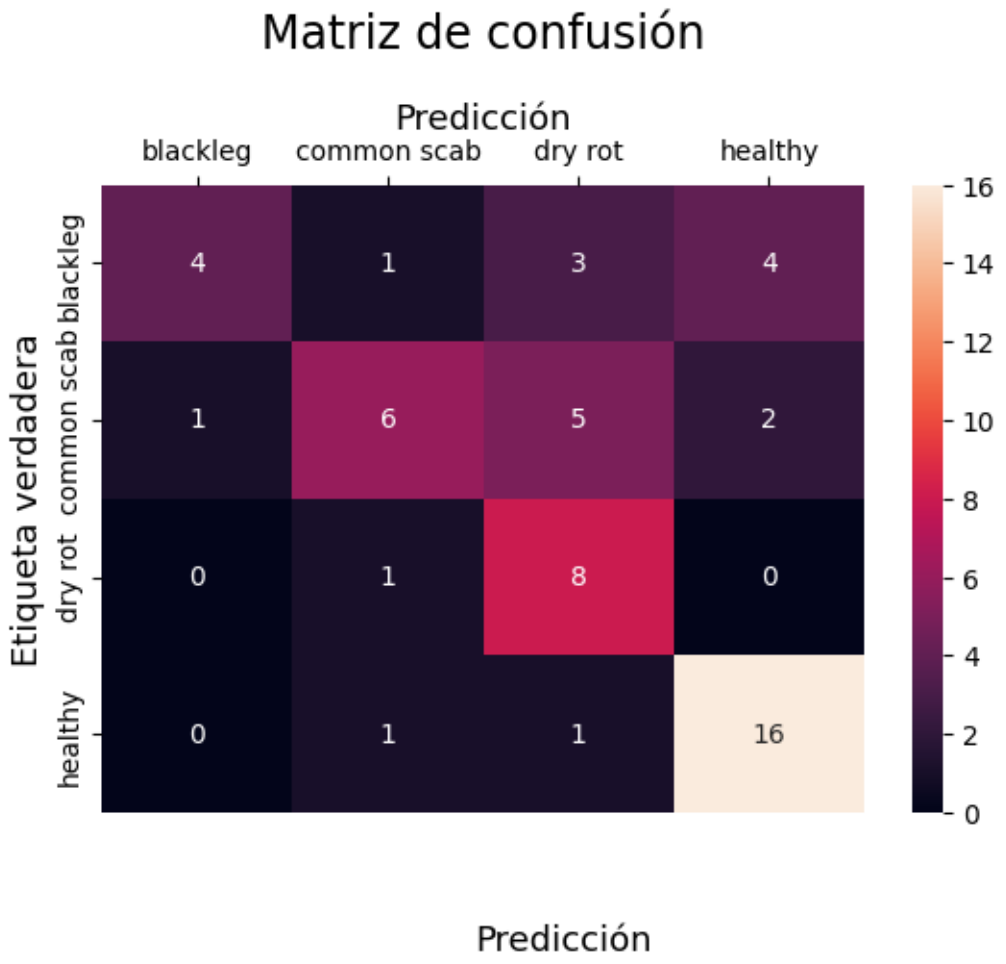
```
Tiempo mínimo mitad ejecución  77.1469989975003 s
Tiempo máximo mitad ejecución:  115.2986666905008
Tiempo promedio mitad ejecución:  85.4239213560999 s
Desviación estándar del tiempo mitad ejecución:  14.945315788865262 s
```

3.12.2 Matriz de confusión y métricas de clasificación correspondientes a las imágenes pre-procesadas con filtro Máximo

[ ]:
```
y_max_preds
```

[ ]:
```
array([2, 3, 3, 2, 2, 2, 2, 3, 1, 2, 0, 3, 3, 3, 3, 1, 3, 2, 3, 0, 2, 2,
       3, 3, 2, 1, 2, 1, 3, 3, 0, 3, 3, 1, 3, 3, 1, 3, 3, 0, 3, 2, 1, 1,
       2, 2, 3, 0, 1, 2, 3, 2, 2])
```

[ ]:
```
cm = confusion_matrix(y_max_test, y_max_preds)
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['blackleg', 'common scab', 'dry rot', 'healthy'],
            yticklabels=['blackleg', 'common scab', 'dry rot', 'healthy'])

plt.ylabel('Etiqueta verdadera', fontsize=13)
plt.title('Matriz de confusión', fontsize=17, pad=20)
plt.gca().xaxis.set_label_position('top')
plt.xlabel('Predicción', fontsize=13)
plt.gca().xaxis.tick_top()

plt.gca().figure.subplots_adjust(bottom=0.2)
plt.gca().figure.text(0.5, 0.05, 'Predicción', ha='center', fontsize=13)
plt.show()
```



Las papas infectadas por blackleg fueron confundidas en un caso como papa infectada por common scab, en tres casos como infectadas por dry rot, y cuatro fueron etiquetadas como papas sanas. En el caso de las papas infectadas por common scab, una fue clasificada como papa infectada por blackleg, cinco fueron clasificadas como papas infectadas por dry rot, y dos papas enfermas son etiquetadas como papas sanas. En el caso de las papas infectadas por dry rot, una fue etiquetada como infectada por common scab. En el caso de las papas sanas, casi todas fueron clasificadas correctamente, siendo una papa clasificada como papa infectada con common scab y otra como infectada por dry rot.

[ ]:
```
print('Accuracy  :', accuracy_score(y_max_test, y_max_preds)*100 ,"%")
print('Precision :' , precision_score(y_max_test, y_max_preds,␣
 ↪average='weighted')*100 ,"%")
print('Recall    :', recall_score(y_max_test, y_max_preds,␣
 ↪average='weighted')*100 ,"%")
print('F1 Score  :',f1_score(y_max_test, y_max_preds, average='weighted')*100␣
 ↪,"%")
```

```
Accuracy  : 64.15094339622641 %
```

```
Precision : 68.41421989035751 %
Recall    : 64.15094339622641 %
F1 Score  : 62.056355071845104 %
```

3.13 Clasificador AdaBoost para dataset filtro Mínimo

La determinación del tiempo de procesamiento del clasificador AdaBoost fue realizada 5 veces, considerando desde la transformación de las imágenes a arrays hasta completar la predicción de las cuatro clases de papas. El tiempo del procesamiento del 50% de las imágenes será calculado dividiendo el tiempo total por dos. Las imágenes del dataset fueron transformadas en arrays 1D mediante embedding directo antes de realizar su clasificación.

[ ]:
```
ex_time_min = []
half_time_min = []

j = 1

while j < 6:
    start = time.perf_counter()

    flat_data_arr_min=[]
    target_arr_min=[]
    datadir_min='/content/drive/MyDrive/potato/data_min'

    for i in Categoría:
        print(f'Cargando categoría: {i}')
        path=os.path.join(datadir_min,i)
        for img in os.listdir(path):
            img_array=imread(os.path.join(path,img))
            img_resized=resize(img_array,(200,200,3))
            flat_data_arr_min.append(img_resized.flatten())
            target_arr_min.append(Categoría.index(i))
    flat_data_min=np.array(flat_data_arr_min)
    target_min=np.array(target_arr_min)
    df_papa_min=pd.DataFrame(flat_data_min)
    df_papa_min['Categoría']=target_min
    df_papa_min

    x_min=df_papa_min.iloc[:,:-1]
    y_min=df_papa_min.iloc[:,-1]
  ␣
↪x_min_train,x_min_test,y_min_train,y_min_test=train_test_split(x_min,y_min,test_size=0.
↪20,random_state=42)

    modeloAdab = AdaBoostClassifier(estimator =␣
↪DecisionTreeClassifier(max_depth=1), n_estimators=50,random_state=123).
↪fit(x_min_train,y_min_train)
    y_min_preds = modeloAdab.predict(x_min_test)
```

```
    end = time.perf_counter()
    total = end - start
    print(f"Tiempo transcurrido: {(total)} s")
    ex_time_min.append(total)
    half_t = (total/2)
    half_time_min.append(half_t)
    j +=1
```

```
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 240.9564445369997 s
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 165.79943667399857 s
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 169.9868647050007 s
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 166.199561636 s
Cargando categoría: blackleg
Cargando categoría: common_scab
Cargando categoría: dry_rot
Cargando categoría: potato
Tiempo transcurrido: 166.08163260999936 s
```

3.13.1 Determinación del tiempo de ejecución para el 50% y 100% de las imágenes clasificadas

[ ]:
```
print("Tiempo mínimo ", min(ex_time_min), 's')
print("Tiempo máximo: ", max(ex_time_min), 's')
print("Tiempo promedio: ", sum(ex_time_min)/len(ex_time_min), 's')
print("Desviación estándar del tiempo: ", np.std(ex_time_min), 's')
```

```
Tiempo mínimo  165.79943667399857 s
Tiempo máximo:  240.9564445369997 s
Tiempo promedio:  181.80478803239967 s
Desviación estándar del tiempo:  29.61585304675923 s
```

[ ]:
```
print("Tiempo mínimo mitad ejecución ", min(half_time_min), 's')
print("Tiempo máximo mitad ejecución: ", max(half_time_min), 's')
print("Tiempo promedio mitad ejecución: ", sum(half_time_min)/
 ↪len(half_time_min), 's')
print("Desviación estándar del tiempo mitad ejecución: ", np.
 ↪std(half_time_min), 's')
```

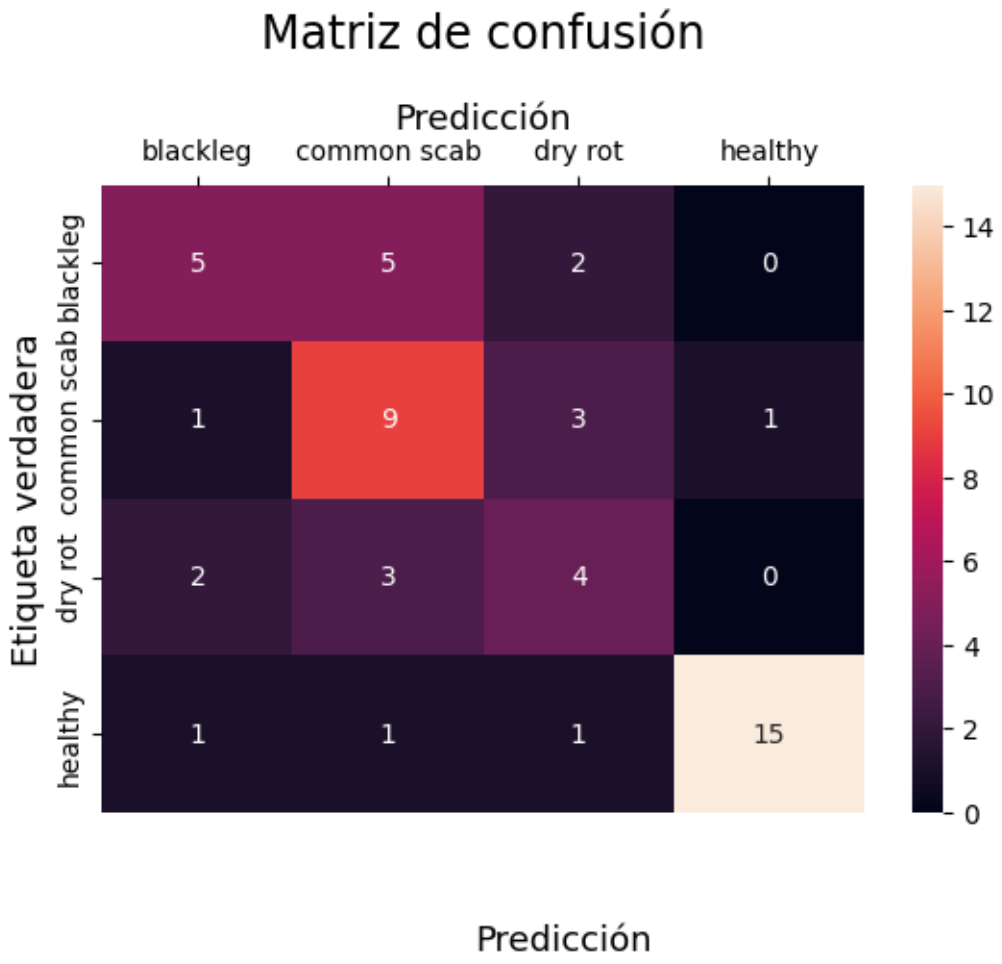
```
Tiempo mínimo mitad ejecución  82.89971833699929 s
Tiempo máximo mitad ejecución:  120.47822226849985 s
Tiempo promedio mitad ejecución:  90.90239401619984 s
Desviación estándar del tiempo mitad ejecución:  14.807926523379615 s
```

3.13.2 Matriz de confusión y métricas de clasificación correspondientes a las imágenes pre-procesadas con filtro Mínimo

[ ]:
```
y_min_preds
```

[ ]:
```
array([3, 3, 3, 2, 1, 2, 1, 2, 1, 1, 0, 3, 1, 3, 3, 3, 1, 2, 3, 1, 1, 0,
       1, 3, 1, 0, 1, 2, 2, 1, 1, 0, 3, 1, 0, 3, 1, 3, 1, 0, 3, 0, 1, 3,
       2, 2, 3, 0, 0, 1, 3, 2, 2])
```

[ ]:
```
cm = confusion_matrix(y_min_test, y_min_preds)
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['blackleg', 'common scab', 'dry rot', 'healthy'],
            yticklabels=['blackleg', 'common scab', 'dry rot', 'healthy'])

plt.ylabel('Etiqueta verdadera', fontsize=13)
plt.title('Matriz de confusión', fontsize=17, pad=20)
plt.gca().xaxis.set_label_position('top')
plt.xlabel('Predicción', fontsize=13)
plt.gca().xaxis.tick_top()

plt.gca().figure.subplots_adjust(bottom=0.2)
plt.gca().figure.text(0.5, 0.05, 'Predicción', ha='center', fontsize=13)
plt.show()
```



Las papas infectadas por blackleg fueron confundidas en cinco casos como papas infectadas por common scab, y en dos casos como infectadas por dry rot. En el caso de las papas infectadas por common scab, una fue clasificada como papa infectada por blackleg, tres fueron clasificadas como papas infectadas por dry rot, y una fue etiquetada como una papa sana. En el caso de las papas infectadas por dry rot, dos fueron clasificadas como papas infectadas por blackleg, y tres como infectadas por common scab. En el caso de las papas sanas, quince fueron clasificadas correctamente, siendo una papa clasificada como papa infectada con blackleg, una como infectada por common scab, y otra como infectada por dry rot.

[ ]:
```
print('Accuracy  :', accuracy_score(y_min_test, y_min_preds)*100 ,"%")
print('Precision :' , precision_score(y_min_test, y_min_preds,␣
 ↪average='weighted')*100 ,"%")
print('Recall    :', recall_score(y_min_test, y_min_preds,␣
 ↪average='weighted')*100 ,"%")
print('F1 Score  :',f1_score(y_min_test, y_min_preds, average='weighted')*100␣
 ↪,"%")
```

```
Accuracy  : 62.264150943396224 %
```

```
Precision : 64.41823899371069 %
Recall    : 62.264150943396224 %
F1 Score  : 62.75681573523152 %
```

3.14 Análisis de resultados del procesamiento de la muestra del dataset de papas

Modelo SVM	Accuracy	Precision	Recall	F1 Score
Modelo 1 (SVM sin filtro)	64.15 %	63.41 %	64.15 %	62.58 %
Modelo 2 (SVM filtro Gaussiano)	56.60 %	61.22 %	56.60 %	57.74 %
Modelo 3 (SVM filtro Sobel)	81.13 %	81.85 %	81.13 %	80.96 %
Modelo 4 (SVM filtro Máximo)	60.38 %	59.30 %	60.38 %	59.35 %
Modelo 5 (SVM filtro Mínimo)	56.60 %	66.51 %	56.60 %	57.54 %

Modelo ResNet50V2	Validation Accuracy	Validation Loss	Test Accuracy	Test Loss
Modelo 6 (ResNet50V2 fine-tuning sin filtro)	84.62 %	0.543	77.78 %	0.651
Modelo 7 (ResNet50V2 fine-tuning filtro Gaussiano)	76.92 %	0.794	51.85 %	1.062
Modelo 8 (ResNet50V2 fine-tuning filtro Sobel)	57.69 %	0.944	51.85 %	1.053
Modelo 9 (ResNet50V2 fine-tuning filtro Máximo)	57.69 %	0.898	55.55 %	0.936

Modelo ResNet50V2	Validation Accuracy	Validation Loss	Test Accuracy	Test Loss
Modelo 10 (ResNet50V2 fine-tuning filtro Mínimo)	65.38 %	0.724	48.14 %	1.010

Modelo ResNet50V2	Accuracy	Precision	Recall	F1 Score
Modelo 6 (ResNet50V2 fine-tuning sin filtro)	78 %	77 %	78 %	77 %
Modelo 7 (ResNet50V2 fine-tuning filtro Gaussiano)	52 %	50 %	52 %	50 %
Modelo 8 (ResNet50V2 fine-tuning filtro Sobel)	52 %	49 %	52 %	50 %
Modelo 9 (ResNet50V2 fine-tuning filtro Máximo)	56 %	54 %	56 %	54 %
Modelo 10 (ResNet50V2 fine-tuning filtro Mínimo)	48 %	44 %	48 %	45 %

Modelo AdaBoost	Accuracy	Precision	Recall	F1 Score
Modelo 11 (AdaBoost sin filtro)	67.92 %	69.40 %	67.92 %	68.45 %
Modelo 12 (AdaBoost filtro Gaussiano)	58.49 %	63.15 %	58.49 %	60.56 %
Modelo 13 (AdaBoost filtro Sobel)	47.17 %	54.62 %	47.17 %	49.59 %

Modelo AdaBoost	Accuracy	Precision	Recall	F1 Score
Modelo 14 (AdaBoost filtro Máximo)	64.15 %	68.41 %	64.15 %	62.06 %
Modelo 15 (AdaBoost filtro Mínimo)	62.26 %	64.42 %	62.26 %	62.76 %

Al comparar los resultados de los procesamientos realizados mediante el algoritmo Support Vector Machine, el modelo de deep learning ResNet50V2, y el algoritmo AdaBoost con árboles de decisión como estimador débil, se confirma que el modelo con mejores resultados es el modelo 3. Se comprueba que en el caso de los otros modelos el filtro Sobel resultó ser inadecuado, debido a que este filtro sólo detecta los bordes de una imagen, suprimiendo los colores de las lesiones de las papas.

En el caso de los modelos ResNet50V2 y AdaBoost clasificaron mejor las fotografías a las cuales no se les aplicó un filtro. Esto implicaría que ambos algoritmos no requieren la aplicación de filtros a las imágenes, siendo incluso perjudicial para su funcionamiento. La disminución del rendimiento al aplicar filtros es especialmente notoria en el caso del modelo ResNet50V2, posiblemente debido a que sus pesos fueron entrenados utilizando el dataset ImageNet, el cual sólo incluye fotografías en color a las cuales no se les aplicó filtros. Además, los filtros alteran las características de las imágenes, dificultando su correcta clasificación.

El mayor problema de todos los modelos ResNet50V2 es el error observado al realizar los procesos de fine-tuning, por lo tanto, es importante verificar a futuro si es posible disminuir dicho valor utilizando datasets de mayor tamaño y mejor calidad de imagen.

3.15 Análisis de los tiempos de procesamiento del 50% y 100% de las imágenes del dataset

Modelo	tiempo mínimo 100% ejecución (s)	tiempo máximo 100% ejecución (s)	tiempo promedio 100% ejecución (s)	desviación estándar tiempo 100% ejecución (s)
Modelo 1 (SVM sin filtro)	14.01	22.52	15.71	2.18
Modelo 2 (SVM filtro Gaussiano)	14.39	18.24	15.72	1.35
Modelo 3 (SVM filtro Sobel)	14.13	29.14	15.71	5.34
Modelo 4 (SVM filtro Máximo)	14.74	34.60	22.04	6.15

Modelo	tiempo mínimo 100% ejecución (s)	tiempo máximo 100% ejecución (s)	tiempo promedio 100% ejecución (s)	desviación estándar tiempo 100% ejecución (s)
Modelo 5 (SVM filtro Mínimo)	14.87	30.03	21.65	5.26
Modelo 6 (ResNet50V2 fine-tuning sin filtro)	11.57	17.27	14.44	2.18
Modelo 7 (ResNet50V2 fine-tuning filtro Gaussiano)	11.60	13.49	12.41	0.52
Modelo 8 (ResNet50V2 fine-tuning filtro Sobel)	11.52	12.93	12.03	5.34
Modelo 9 (ResNet50V2 fine-tuning filtro Máximo)	11.45	14.35	12.64	1.17
Modelo 10 (ResNet50V2 fine-tuning filtro Mínimo)	11.74	12.46	11.71	0.65
Modelo 11 (AdaBoost sin filtro)	192.49	265.19	222.63	24.06
Modelo 12 (AdaBoost filtro Gaussiano)	171.96	201.40	178.49	11.47
Modelo 13 (AdaBoost filtro Sobel)	148.84	223.22	222.63	29.46
Modelo 14 (AdaBoost filtro Máximo)	154.29	230.60	170.85	29.89
Modelo 15 (AdaBoost filtro Mínimo)	165.80	240.96	181.80	29.65

Modelo	tiempo mínimo 50% ejecución (s)	tiempo máximo 50% ejecución (s)	tiempo promedio 50% ejecución (s)	desviación estándar tiempo 50% ejecución (s)
Modelo 1 (SVM sin filtro)	7.01	11.26	7.85	1.09
Modelo 2 (SVM filtro Gaussiano)	7.19	9.12	7.86	0.68
Modelo 3 (SVM filtro Sobel)	7.06	14.57	9.55	2.67
Modelo 4 (SVM filtro Máximo)	7.37	17.30	10.83	3.08
Modelo 5 (SVM filtro Mínimo)	7.44	15.01	8.54	2.63
Modelo 6 (ResNet50V2 fine-tuning sin filtro)	5.79	8.63	7.22	1.09
Modelo 7 (ResNet50V2 fine-tuning filtro Gaussiano)	5.80	6.75	6.21	0.38
Modelo 8 (ResNet50V2 fine-tuning filtro Sobel)	5.76	6.47	6.02	0.26
Modelo 9 (ResNet50V2 fine-tuning filtro Máximo)	5.72	7.18	6.32	0.59
Modelo 10 (ResNet50V2 fine-tuning filtro Mínimo)	5.87	6.75	6.23	0.33
Modelo 11 (AdaBoost sin filtro)	96.24	132.60	111.32	2.01
Modelo 12 (AdaBoost filtro Gaussiano)	85.98	100.70	89.25	5.73

Modelo	tiempo mínimo 50% ejecución (s)	tiempo máximo 50% ejecución (s)	tiempo promedio 50% ejecución (s)	desviación estándar tiempo 50% ejecución (s)
Modelo 13 (AdaBoost filtro Sobel)	74.42	111.61	82.17	14.73
Modelo 14 (AdaBoost filtro Máximo)	77.15	115.30	85.42	14.94
Modelo 15 (AdaBoost filtro Mínimo)	82.90	120.48	90.90	14.81

Los modelos más rápidos son los cinco modelos ResNet50V2, pero sólo ocurre esto debido a que sus procesos de fine-tuning fueron realizados mediante el uso de una GPU Nvidia T4. Estos reultados serían muy distintos si dichos modelos hubieran sido entrenados usando una CPU. Además, el uso de modelos de deep learning requiere instalar librerías como Tensorflow o Pytorch, las cuales ocupan gigabytes de espacio en disco, lo cual no es viable para crear aplicaciones móviles.

Respecto a los modelos entrenados con CPU, todos los modelos SVM son muchísimos mas rápidos que los modelos AdaBoost con cincuenta estimadores débiles, pero su desventaja principal es su alto consumo de memoria RAM.

4 Conclusiones

Los modelos estudiados en general son capaces de distinguir las papas sanas de las papas infectadas por algún tipo de hongo o bacteria. Sin embargo, es más complejo poder diferenciar entre las papas infectadas por blackleg, common scab y dry rot.

El uso de filtro Sobel solamente es recomendable para el caso del algoritmo Support Vector Machines. Los modelos ResNetV2 y AdaBoost con árboles de decisión como estimadores débiles no requieren este tipo de pre-procesamiento.

La velocidad de procesamiento de las imágenes no es realmente comparable debido a que los modelos ResNet50V2 fueron entrenados mediante una GPU, mientras que los modelos SVM y AdaBoost fueron entrenados con una CPU.

El tamaño del dataset depende del algoritmo a entrenar, dado que el algoritmo SVM funciona adecuadamente con un dataset de 262 imágenes, pero los modelos de deep learning requieren datasets con una cantidad mucho mayor de fotografías para obtener buenos resultados.

Los modelos de deep learning sólo pueden ser utilizados en computadores debido al tamaño de las librerías que requieren para funcionar y el tipo de GPU necesario para entrenarlos en un tiempo razonable, siendo los algoritmos de machine learning clásico más adecuados para desarrollar aplicaciones móviles.