p-1001-riceleaf-disease-detection

June 6, 2024

1 Problem Statement

1.1 Task 1:-Prepare a complete data analysis report on the given data.

1.2 Task 2:-Create a model which can classify the three major attacking diseases of rice plants like leaf blast, bacterial blight and brown spot.

1.3 Task3:- Analyze various techniques like Data Augmentation, etc and create a report on that.

1.4 Domain Expertise

- The dataset includes 119 jpg images of rice leaves effected by different types of diseases.
- The classes are:-
- **Bacterial Leaf Blight**
- Bacterial leaf blight of rice (BB) is a disease caused by the Gram-negative bacterium Xanthomonas oryzae pv.Bacterial leaf blight is often first noticed in fields as brown areas about 3 to 4 feet in diameter. Leaf symptoms appear as irregular brown spots, often beginning on the leaf margins. Lesions initially have an irregular yellow halo and may appear watersoaked.
- **Brown Spot**
- Brown spot is a fungal disease that can infect both seedlings and mature plants. The disease causes blight on seedlings, which are grown from heavily infected seeds, and can cause 10-58% seedling mortality.
- **Leaf Smut**
- Leaf smut, caused by the fungus Entyloma oryzae, is a widely distributed, but somewhat minor, disease of rice.The fungus produces slightly raised, angular, black spots (sori) on both sides of the leaves.The black spots are about 0.5 to 5.0 millimeters long and 0.5 to 1.5 millimeters wide. **The class Leaf Smut contains 39 jpg images while rest all classes contain 40 jpg images.**

1.5 Importing Necessary Libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
```

[2]:
```
input_shape_2D = (224,224)
input_shape_3D = (224,224,3)
seed = 1
batch_size = 5
epochs = 30
```

[3]:
```
import os

# Create directory if it doesn't exist
directory_path = '/content/drive/My Drive/content/drive'
if not os.path.exists(directory_path):
    os.makedirs(directory_path)
else:
    print(f"The directory '{directory_path}' already exists.")
```

[4]:
```
!pip install -U -q PyDrive
```

[5]:
```
import shutil

# Unmount the directory
shutil.rmtree('/content/drive')
```

[6]:
```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

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

1.6 Loading the Data

[7]:
```
data = tf.keras.utils.image_dataset_from_directory(directory="/content/drive/
 ↪MyDrive/PRCP-1001-RiceLeaf",
                                                    labels='inferred',
                                                    label_mode='int',
                                                    class_names=None,
                                                    color_mode='rgb',
                                                    image_size=input_shape_2D,
                                                    seed=seed)
```

```
Found 119 files belonging to 1 classes.
```

[8]:
```
# print class names

class_names = data.class_names
class_names
```

[8]:
```
['Data']
```

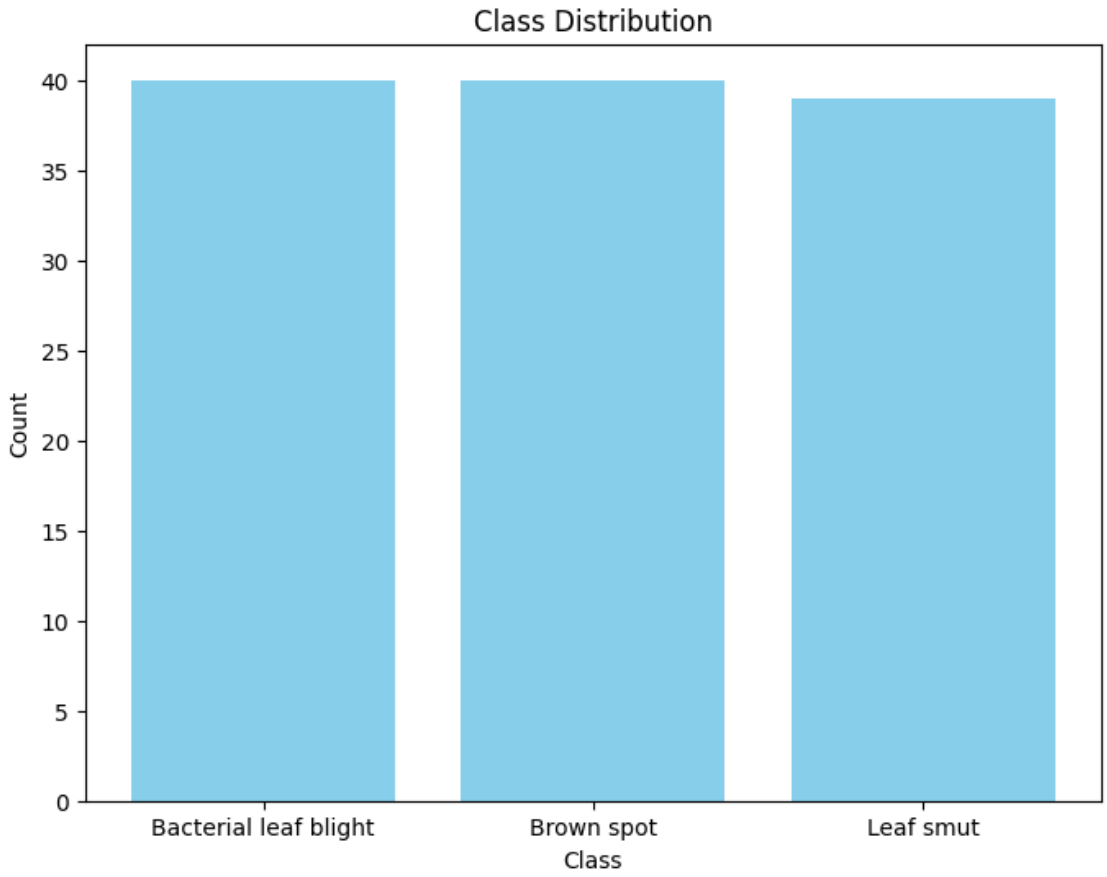
The 3 Classes are - Bacterial leaf blight, Brown spot, Leaf smut.

[9]:
```
# Initialize an empty dictionary to store class counts
class_counts = {}

# Iterate through the dataset to count occurrences of each class label
for images, labels in data:
    for label in labels:
        class_counts[label.numpy()] = class_counts.get(label.numpy(), 0) + 1

# Print the class counts
print(class_counts)
```

```
{0: 119}
```

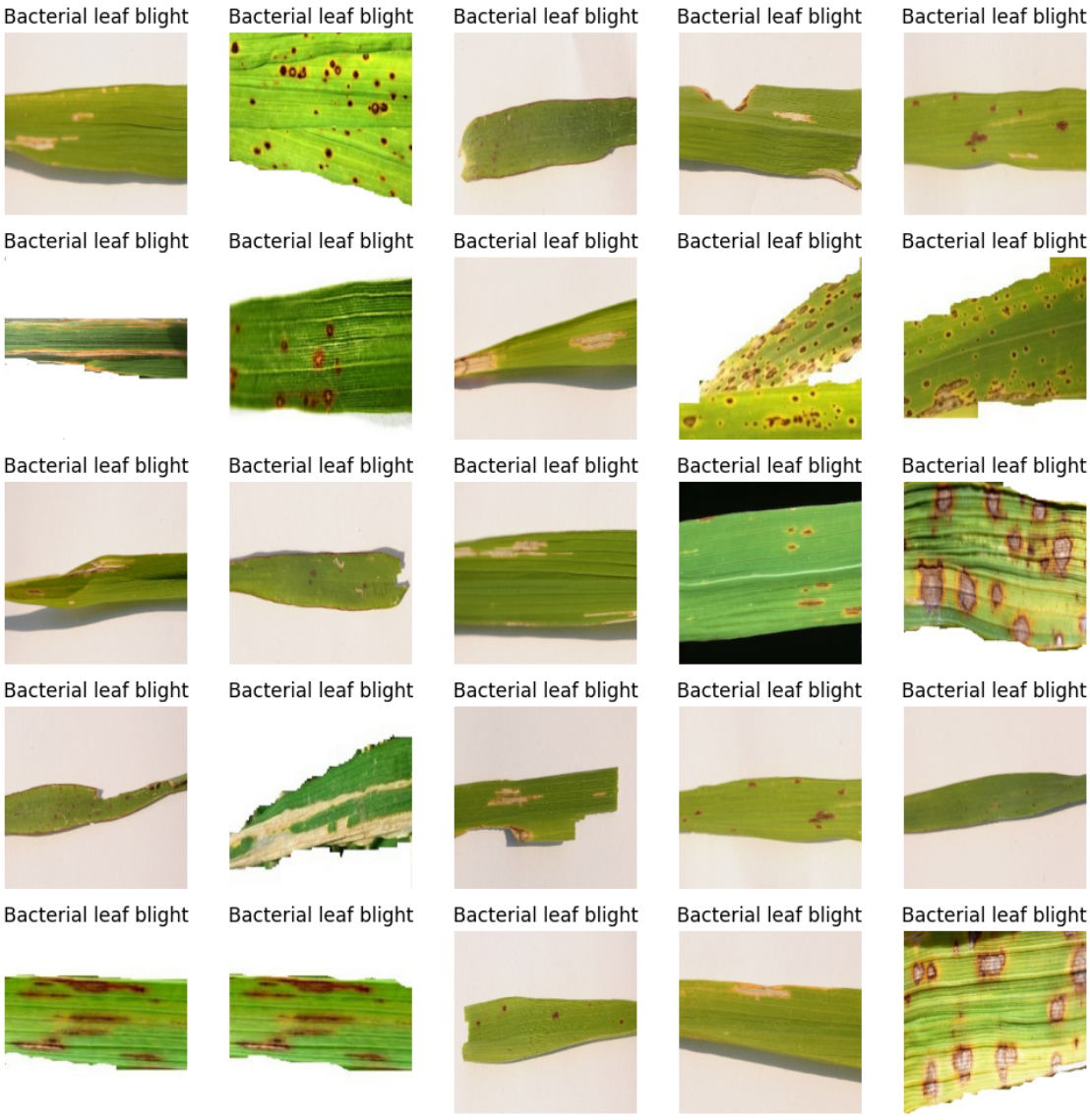
1.7 Image Visualisation

[10]:
```
# Class names and counts
class_names = ['Bacterial leaf blight', 'Brown spot', 'Leaf smut']
class_counts = [40, 40, 39]  # Replace with your actual counts

# Plotting
plt.figure(figsize=(8, 6))
plt.bar(class_names, class_counts, color='skyblue')
plt.xlabel('Class')
plt.ylabel('Count')
plt.title('Class Distribution')
plt.show()
```



- From the graph we can clearly see that there are a total of 119 images.
- 40 in Bacterial Leaf Blight and Brown Spot.
- 39 images available in Leaf Smut.

[11]:
```
plt.figure(figsize=(10,10))
for images , labels in data.take(1):
    for i in range(25):
        plt.subplot(5,5,i+1)
        plt.imshow(images[i].numpy().astype('uint8'))
        plt.title(class_names[labels[i]])
        plt.axis('off')
    plt.tight_layout()
```



1.7.1 These are the inputs of our model.

[12]:
```
# Create Dependent(y) and Independent(X) variable

X = []
y = []

for images , labels in data:
    X.append(images.numpy())
    y.append(labels.numpy())
```

```
# convert X , y list into numpy array

X = np.concatenate(X ,axis=0)
y = np.concatenate(y ,axis=0)
```

[13]:
```
X
```

[13]:
```
array([[[[  4.761161 ,   9.761161 ,   3.7611609],
         [  5.566964 ,   9.716518 ,   4.       ],
         [  9.417412 ,  10.611608 ,   5.6116076],
         …,
         [  6.       ,  11.       ,   5.       ],
         [  8.       ,  10.       ,   5.       ],
         [  8.       ,  10.       ,   5.       ]],

        [[  4.761161 ,   9.761161 ,   3.7611609],
         [  5.566964 ,   9.716518 ,   4.       ],
         [  9.417412 ,  10.611608 ,   5.6116076],
         …,
         [  6.       ,  11.       ,   5.       ],
         [  8.       ,  10.       ,   5.       ],
         [  8.       ,  10.       ,   5.       ]],

        [[  4.761161 ,   9.761161 ,   3.7611609],
         [  5.566964 ,   9.716518 ,   4.       ],
         [  9.417412 ,  10.611608 ,   5.6116076],
         …,
         [  6.       ,  11.       ,   5.       ],
         [  8.       ,  10.       ,   5.       ],
         [  8.       ,  10.       ,   5.       ]],

        …,

        [[ 10.238839 ,  11.238839 ,   6.238839 ],
         [  9.283482 ,  10.283482 ,   5.283482 ],
         [  9.       ,  10.       ,   5.       ],
         …,
         [  4.       ,   9.       ,   5.       ],
         [  4.       ,   9.       ,   3.       ],
         [  4.       ,   9.       ,   3.       ]],

        [[ 10.238839 ,  11.238839 ,   6.238839 ],
         [  9.283482 ,  10.283482 ,   5.283482 ],
         [  9.       ,  10.       ,   5.       ],
         …,
         [  4.       ,   9.       ,   5.       ],
         [  4.       ,   9.       ,   3.       ],
```

```
  [  4.       ,   9.       ,   3.       ]],

 [[ 10.472878 ,  11.472878 ,   6.4728785],
  [  9.005697 ,  10.005697 ,   5.005697 ],
  [  9.979904 ,  10.979904 ,   5.979904 ],
  …,
  [  3.0200958,   8.020096 ,   4.020096 ],
  [  4.979904 ,   9.979904 ,   3.9799042],
  [  4.979904 ,   9.979904 ,   3.9799042]]],


[[[216.66805  , 174.85745  ,  72.54496  ],
  [197.44548  , 161.2229   ,  33.771362 ],
  [170.22768  , 134.66805  ,  25.138393 ],
  …,
  [255.       , 255.       , 255.       ],
  [255.       , 255.       , 255.       ],
  [255.       , 255.       , 255.       ]],

 [[179.37819  , 140.31888  ,  31.386158 ],
  [160.28348  , 126.27232  ,   8.291773 ],
  [134.26115  , 104.29177  ,   6.8635206],
  …,
  [255.       , 255.       , 255.       ],
  [255.       , 255.       , 255.       ],
  [255.       , 255.       , 255.       ]],

 [[154.14572  , 125.19483  ,  11.570471 ],
  [142.1738   , 116.95918  ,   5.7334166],
  [130.93431  , 111.26913  ,  22.705677 ],
  …,
  [255.       , 255.       , 255.       ],
  [255.       , 255.       , 255.       ],
  [255.       , 255.       , 255.       ]],

 …,

 [[255.       , 255.       , 255.       ],
  [255.       , 255.       , 255.       ],
  [255.       , 255.       , 255.       ],
  …,
  [255.       , 255.       , 255.       ],
  [255.       , 255.       , 255.       ],
  [255.       , 255.       , 255.       ]],

 [[255.       , 255.       , 255.       ],
  [255.       , 255.       , 255.       ],
```

```
  [255.       , 255.       , 255.       ],
  …,
  [255.       , 255.       , 255.       ],
  [255.       , 255.       , 255.       ],
  [255.       , 255.       , 255.       ]],

 [[255.       , 255.       , 255.       ],
  [255.       , 255.       , 255.       ],
  [255.       , 255.       , 255.       ],
  …,
  [255.       , 255.       , 255.       ],
  [255.       , 255.       , 255.       ],
  [255.       , 255.       , 255.       ]]],


[[[236.50223  , 227.50223  , 220.50223  ],
  [237.       , 226.       , 220.       ],
  [237.       , 226.       , 220.       ],
  …,
  [233.33133  , 223.32686  , 216.82909  ],
  [234.36594  , 223.43155  , 217.23468  ],
  [235.74338  , 221.74783  , 217.24113  ]],

 [[235.5067   , 224.5067   , 218.5067   ],
  [237.8683   , 226.8683   , 221.8817   ],
  [236.05615  , 225.05615  , 219.05615  ],
  …,
  [232.94388  , 221.94388  , 215.94388  ],
  [231.36147  , 220.36147  , 214.49507  ],
  [235.30724  , 222.30724  , 216.30724  ]],

 [[236.19283  , 225.19283  , 219.19283  ],
  [237.44385  , 226.44385  , 221.42152  ],
  [236.       , 225.       , 219.       ],
  …,
  [234.94438  , 223.94438  , 217.94438  ],
  [234.51115  , 221.44672  , 215.64006  ],
  [232.09775  , 220.48885  , 214.48885  ]],

 …,

 [[237.1844   , 224.1844   , 216.1844   ],
  [237.57555  , 224.57555  , 216.57555  ],
  [237.       , 224.       , 216.       ],
  …,
  [231.88623  , 220.88623  , 214.88623  ],
  [232.0674   , 221.0674   , 215.0674   ],
```

```
  [229.86603  , 218.86603  , 212.86603  ]],

 [[235.37723  , 225.37723  , 216.37723  ],
  [235.56001  , 222.56001  , 214.56001  ],
  [237.       , 224.       , 216.       ],
  …,
  [234.44907  , 221.44907  , 215.44907  ],
  [230.       , 219.       , 213.       ],
  [232.       , 219.       , 213.       ]],

 [[235.       , 225.       , 216.       ],
  [234.56781  , 223.56781  , 217.56781  ],
  [235.       , 225.       , 216.       ],
  …,
  [233.88623  , 218.88623  , 213.88623  ],
  [232.       , 219.       , 213.       ],
  [233.81226  , 220.81226  , 214.81226  ]]],


…,


[[[233.       , 224.       , 219.       ],
  [235.       , 224.       , 220.       ],
  [236.49777  , 225.49777  , 221.49777  ],
  …,
  [231.99553  , 222.99553  , 215.99553  ],
  [234.43542  , 223.30418  , 217.30418  ],
  [235.       , 225.50223  , 221.99553  ]],

 [[233.31555  , 222.31555  , 218.31555  ],
  [235.1317   , 224.1317   , 220.1317   ],
  [236.05768  , 225.05768  , 221.05768  ],
  …,
  [234.11377  , 223.11377  , 217.11377  ],
  [235.50493  , 226.37486  , 219.4399   ],
  [235.80887  , 226.80887  , 220.79549  ]],

 [[233.       , 222.       , 218.       ],
  [235.       , 224.       , 220.       ],
  [236.       , 225.       , 221.       ],
  …,
  [235.48885  , 224.48885  , 218.48885  ],
  [237.4244   , 228.4244   , 221.4244   ],
  [235.81561  , 226.81561  , 220.83794  ]],

 …,
```

```
 [[[235.19283  , 222.19283  , 213.19283  ],
  [233.       , 219.       , 210.       ],
  [232.49138  , 219.49138  , 210.49138  ],
  …,
  [231.94438  , 220.94438  , 214.94438  ],
  [231.62067  , 221.62067  , 212.62067  ],
  [232.68164  , 221.68164  , 215.68164  ]],

 [[233.81392  , 220.81392  , 211.81392  ],
  [234.8683   , 220.8683   , 211.8683   ],
  [233.06958  , 220.06958  , 211.06958  ],
  …,
  [232.43564  , 221.43564  , 215.43564  ],
  [232.86816  , 219.86816  , 213.86816  ],
  [237.12952  , 224.12952  , 216.12952  ]],

 [[235.19398  , 222.19398  , 213.19398  ],
  [233.49834  , 220.49834  , 211.49834  ],
  [230.94334  , 220.94334  , 210.94334  ],
  …,
  [231.88623  , 220.88623  , 214.88623  ],
  [234.43213  , 219.43213  , 214.43213  ],
  [236.       , 222.2456   , 214.6228   ]]],


[[[242.       , 237.       , 233.       ],
  [242.06615  , 237.06615  , 233.06615  ],
  [244.       , 236.       , 234.       ],
  …,
  [236.4451   , 225.4451   , 219.4451   ],
  [237.86816  , 229.86816  , 218.86816  ],
  [239.3772   , 229.6228   , 223.       ]],

 [[241.       , 236.       , 232.       ],
  [241.1317   , 236.1317   , 232.1317   ],
  [241.94385  , 236.94385  , 233.94385  ],
  …,
  [238.4933   , 227.4933   , 221.4933   ],
  [238.       , 229.06503  , 219.93497  ],
  [238.38554  , 228.63115  , 223.39388  ]],

 [[241.       , 236.       , 232.       ],
  [241.55615  , 236.55615  , 232.55615  ],
  [240.88617  , 235.88617  , 232.88617  ],
  …,
  [238.       , 227.       , 221.       ],
```

```
  [238.       , 229.       , 220.88754  ],
  [237.6228   , 227.9861   , 222.66774  ]],

 …,

 [[242.       , 234.       , 231.       ],
  [241.       , 233.       , 230.       ],
  [241.88617  , 231.88617  , 229.88617  ],
  …,
  [239.11377  , 229.11377  , 219.11377  ],
  [238.86816  , 229.       , 220.       ],
  [237.51117  , 227.51117  , 218.51117  ]],

 [[241.       , 233.       , 230.       ],
  [241.       , 233.       , 230.       ],
  [241.       , 233.       , 230.       ],
  …,
  [239.       , 229.       , 220.       ],
  [237.42825  , 227.56009  , 218.56009  ],
  [238.49329  , 228.49329  , 219.49329  ]],

 [[242.62277  , 234.62277  , 231.62277  ],
  [240.8683   , 232.8683   , 229.8683   ],
  [238.       , 233.       , 229.       ],
  …,
  [238.       , 228.       , 219.       ],
  [237.       , 228.       , 219.       ],
  [237.       , 228.       , 219.       ]]],


[[[237.62277  , 226.62277  , 220.62277  ],
  [237.49777  , 224.49777  , 218.49777  ],
  [235.8822   , 224.5589   , 214.88667  ],
  …,
  [239.5549   , 231.5549   , 229.5549   ],
  [239.13184  , 231.86816  , 226.       ],
  [239.       , 232.       , 226.       ]],

 [[236.62277  , 225.62277  , 219.62277  ],
  [236.4933   , 226.4933   , 217.4933   ],
  [236.44902  , 226.56438  , 216.5067   ],
  …,
  [239.       , 231.       , 228.       ],
  [239.       , 231.4933   , 227.0134   ],
  [239.5067   , 231.4933   , 226.       ]],

 [[236.       , 225.       , 219.       ],
```

```
       [236.       , 226.       , 217.       ],
       [236.       , 226.       , 216.51115  ],
       …,
       [238.43323  , 231.43323  , 225.43323  ],
       [238.48885  , 231.       , 226.46652  ],
       [238.48885  , 230.51115  , 223.97768  ]],

      …,

      [[237.48883  , 224.48883  , 216.48883  ],
       [238.       , 225.       , 217.       ],
       [234.48883  , 225.48883  , 216.48883  ],
       …,
       [238.51117  , 228.51117  , 218.51117  ],
       [239.       , 231.       , 220.       ],
       [236.       , 226.       , 216.       ]],

      [[237.       , 224.       , 216.       ],
       [238.       , 225.       , 217.       ],
       [236.       , 226.       , 217.       ],
       …,
       [237.55093  , 227.55093  , 218.55093  ],
       [238.42825  , 230.29642  , 219.56009  ],
       [238.       , 228.       , 218.       ]],

      [[236.81224  , 223.81224  , 215.81224  ],
       [238.       , 225.       , 217.       ],
       [238.05717  , 225.05717  , 217.05717  ],
       …,
       [239.88623  , 229.88623  , 220.88623  ],
       [237.50226  , 229.50226  , 218.50226  ],
       [239.       , 229.       , 220.       ]]], dtype=float32)
```

[14]:
```
X = X.astype('float32')/255
```

1.8 Spliting Dataset into Train and Test/Validation data

[15]:
```
# Splitt data into train and test / validation

X_train, X_test = X[:95 ],X[95:]
y_train, y_test = y[:95] ,y[95:]
```

[16]:
```
X_train.shape , X_test.shape
```

[16]:
```
((95, 224, 224, 3), (24, 224, 224, 3))
```

[17]:
```
# Convert labels to one-hot encoding

from tensorflow.keras.utils import to_categorical

y_train = to_categorical(y_train ,len(class_names))
y_test = to_categorical(y_test ,len(class_names))
```

1.9 Model Creation

[18]:
```
from keras.models import Sequential
from keras.layers import Conv2D , MaxPooling2D , Flatten ,Dense ,Dropout
```

[19]:
```
# relu - the usage of ReLU helps to prevent the exponential growth in the␣
 ↪computation required to operate the neural network.
        #It has a derivative of either 0 or 1, depending on whether its input␣
 ↪is negative or not
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(3, activation='softmax')
])
```

[20]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

[21]:
```
history = model.fit(X_train, y_train, epochs=30, batch_size=32,␣
 ↪validation_split=0.2)
```

```
Epoch 1/30
3/3 [==============================] - 11s 3s/step - loss: 0.3903 - accuracy:
0.9605 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 2/30
3/3 [==============================] - 7s 2s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 3/30
3/3 [==============================] - 9s 3s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 4/30
3/3 [==============================] - 7s 2s/step - loss: 0.0000e+00 - accuracy:
```

```
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 5/30
3/3 [==============================] - 9s 2s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 6/30
3/3 [==============================] - 8s 3s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 7/30
3/3 [==============================] - 8s 2s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 8/30
3/3 [==============================] - 9s 3s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 9/30
3/3 [==============================] - 7s 2s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 10/30
3/3 [==============================] - 9s 3s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 11/30
3/3 [==============================] - 7s 2s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 12/30
3/3 [==============================] - 11s 3s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 13/30
3/3 [==============================] - 7s 2s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 14/30
3/3 [==============================] - 9s 2s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 15/30
3/3 [==============================] - 8s 3s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 16/30
3/3 [==============================] - 8s 2s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 17/30
3/3 [==============================] - 9s 3s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 18/30
3/3 [==============================] - 7s 2s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 19/30
3/3 [==============================] - 9s 3s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 20/30
3/3 [==============================] - 7s 2s/step - loss: 0.0000e+00 - accuracy:
```

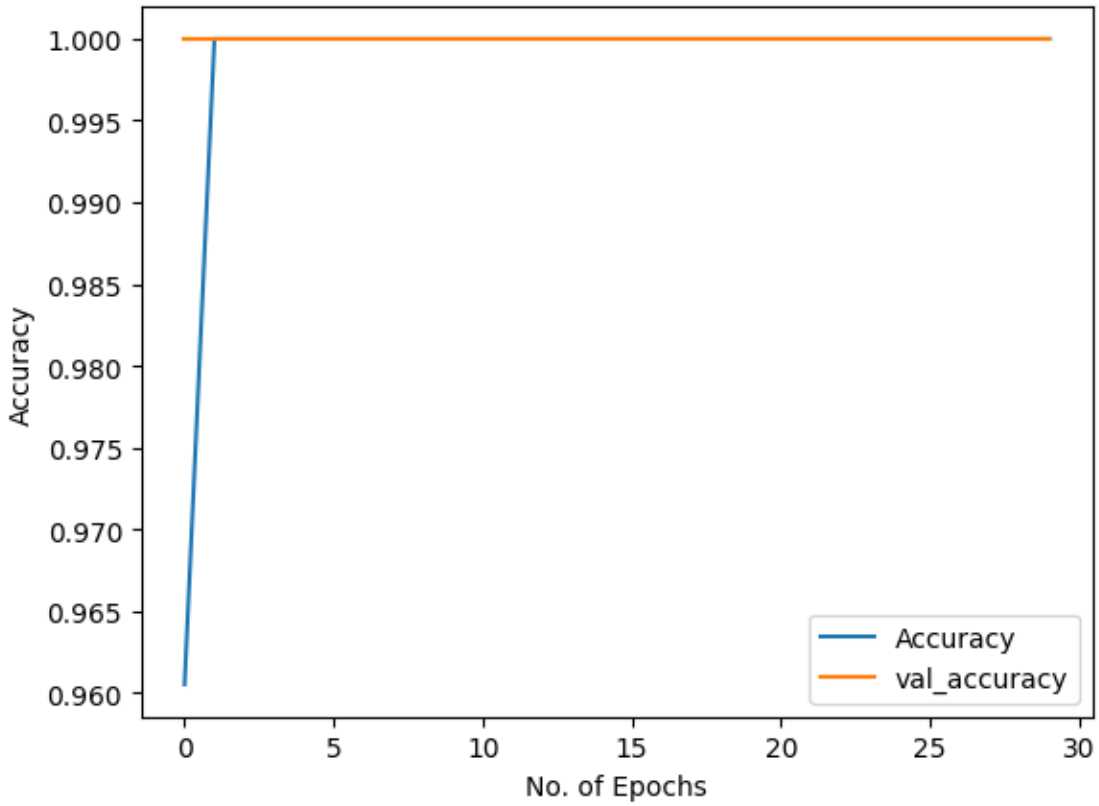
```
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 21/30
3/3 [==============================] - 9s 3s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 22/30
3/3 [==============================] - 8s 3s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 23/30
3/3 [==============================] - 9s 2s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 24/30
3/3 [==============================] - 9s 3s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 25/30
3/3 [==============================] - 7s 2s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 26/30
3/3 [==============================] - 9s 3s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 27/30
3/3 [==============================] - 7s 2s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 28/30
3/3 [==============================] - 9s 3s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 29/30
3/3 [==============================] - 7s 2s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 30/30
3/3 [==============================] - 9s 2s/step - loss: 0.0000e+00 - accuracy:
1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
```

[22]:
```
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test accuracy: {test_accuracy:.4f}")
```

```
1/1 [==============================] - 1s 578ms/step - loss: 0.0000e+00 -
accuracy: 1.0000
Test accuracy: 1.0000
```

[23]:
```
plt.plot(history.history['accuracy'],label='Accuracy')
plt.plot(history.history['val_accuracy'] ,label='val_accuracy')
plt.xlabel("No. of Epochs")
plt.ylabel("Accuracy")

plt.legend(loc = "lower right")
```

[23]:
```
<matplotlib.legend.Legend at 0x7acb38a8fbe0>
```



1.9.1 As the model shows only 70% accuracy we are doing Data Augmentation to improve the accuracy.

1.10 Data Augmentation

[24]:
```
from keras.preprocessing.image import ImageDataGenerator
```

[25]:
```
datagen = ImageDataGenerator(
    rotation_range=45,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

1.11 Model Creation using different Optimizers after Data Augmentation

1.11.1 1.Adam optimizer

[26]:
```
def create_model(input_shape, num_classes):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(256, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')  # Output layer
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
    return model
```

[27]:
```
input_shape = (224, 224, 3)
num_classes = 3
model_da = create_model(input_shape, num_classes)
```

[28]:
```
augmented_data = datagen.flow(X_train, y_train, batch_size=32)
```

[29]:
```
history = model_da.fit(augmented_data, epochs=50,steps_per_epoch=len(X_train) //
 ↪ 25, validation_data=(X_test, y_test))
```

```
Epoch 1/50
3/3 [==============================] - 15s 4s/step - loss: 0.3634 - accuracy:
0.7263 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 2/50
3/3 [==============================] - 13s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 3/50
3/3 [==============================] - 14s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 4/50
3/3 [==============================] - 14s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 5/50
3/3 [==============================] - 14s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 6/50
```

```
3/3 [==============================] - 14s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 7/50
3/3 [==============================] - 15s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 8/50
3/3 [==============================] - 14s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 9/50
3/3 [==============================] - 14s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 10/50
3/3 [==============================] - 13s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 11/50
3/3 [==============================] - 13s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 12/50
3/3 [==============================] - 13s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 13/50
3/3 [==============================] - 14s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 14/50
3/3 [==============================] - 14s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 15/50
3/3 [==============================] - 14s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 16/50
3/3 [==============================] - 13s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 17/50
3/3 [==============================] - 13s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 18/50
3/3 [==============================] - 13s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 19/50
3/3 [==============================] - 14s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 20/50
3/3 [==============================] - 14s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 21/50
3/3 [==============================] - 14s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 22/50
```

```
3/3 [==============================] - 13s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 23/50
3/3 [==============================] - 13s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 24/50
3/3 [==============================] - 13s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 25/50
3/3 [==============================] - 13s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 26/50
3/3 [==============================] - 15s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 27/50
3/3 [==============================] - 14s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 28/50
3/3 [==============================] - 13s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 29/50
3/3 [==============================] - 14s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 30/50
3/3 [==============================] - 14s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 31/50
3/3 [==============================] - 13s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 32/50
3/3 [==============================] - 13s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 33/50
3/3 [==============================] - 14s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 34/50
3/3 [==============================] - 13s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 35/50
3/3 [==============================] - 12s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 36/50
3/3 [==============================] - 14s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 37/50
3/3 [==============================] - 13s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 38/50
```

```
3/3 [==============================] - 14s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 39/50
3/3 [==============================] - 13s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 40/50
3/3 [==============================] - 13s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 41/50
3/3 [==============================] - 14s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 42/50
3/3 [==============================] - 14s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 43/50
3/3 [==============================] - 12s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 44/50
3/3 [==============================] - 14s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 45/50
3/3 [==============================] - 14s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 46/50
3/3 [==============================] - 14s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 47/50
3/3 [==============================] - 13s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 48/50
3/3 [==============================] - 12s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 49/50
3/3 [==============================] - 13s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 50/50
3/3 [==============================] - 14s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
```

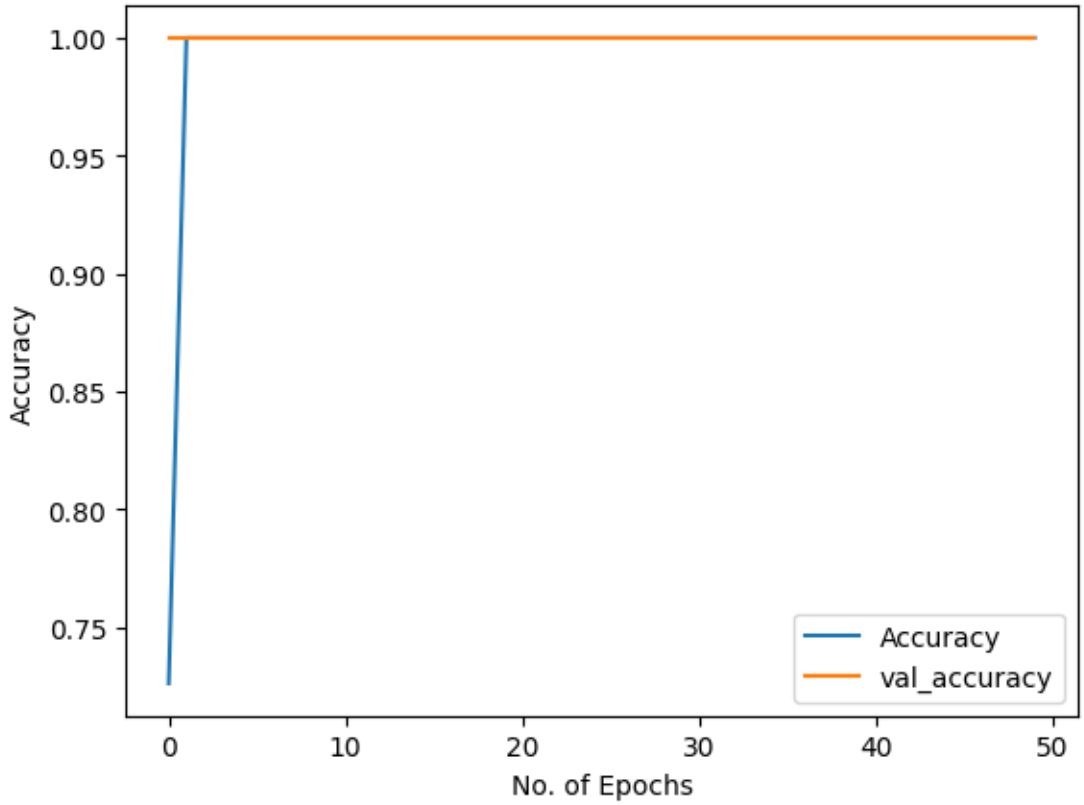
[30]:
```
test_loss, test_accuracy = model_da.evaluate(X_test, y_test)
print(f"Test accuracy: {test_accuracy:.4f}")
```

```
1/1 [==============================] - 1s 720ms/step - loss: 0.0000e+00 -
accuracy: 1.0000
Test accuracy: 1.0000
```

1.11.2 Model shows an accuracy of 87.50 % for Adam Optimizer.

[31]:
```
#Ploting the Accuracy and Validation Accuracy
plt.plot(history.history['accuracy'],label='Accuracy')
plt.plot(history.history['val_accuracy'] ,label='val_accuracy')
plt.xlabel("No. of Epochs")
plt.ylabel("Accuracy")

plt.legend(loc = "lower right")
```

[31]:
```
<matplotlib.legend.Legend at 0x7acb39a4ee60>
```



1.11.3 2.RMSProp Optimizer

[32]:
```
def create_model(input_shape, num_classes):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
```

```
        MaxPooling2D((2, 2)),
        Conv2D(256, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')  # Output layer
    ])
    model.compile(optimizer='RMSprop', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
    return model
```

[33]:
```
input_shape = (224, 224, 3)
num_classes = 3
model_rms = create_model(input_shape, num_classes)
```

[34]:
```
augmented_data = datagen.flow(X_train, y_train, batch_size=32)
```

[35]:
```
history = model_rms.fit(augmented_data, epochs=50,steps_per_epoch=len(X_train) /
 ↪/ 25, validation_data=(X_test, y_test))
```

```
Epoch 1/50
3/3 [==============================] - 15s 5s/step - loss: 0.3548 - accuracy:
0.9158 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 2/50
3/3 [==============================] - 14s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 3/50
3/3 [==============================] - 14s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 4/50
3/3 [==============================] - 14s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 5/50
3/3 [==============================] - 13s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 6/50
3/3 [==============================] - 12s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 7/50
3/3 [==============================] - 12s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 8/50
3/3 [==============================] - 14s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 9/50
3/3 [==============================] - 14s 4s/step - loss: 0.0000e+00 -
```

```
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 10/50
3/3 [==============================] - 15s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 11/50
3/3 [==============================] - 14s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 12/50
3/3 [==============================] - 14s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 13/50
3/3 [==============================] - 14s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 14/50
3/3 [==============================] - 14s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 15/50
3/3 [==============================] - 14s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 16/50
3/3 [==============================] - 13s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 17/50
3/3 [==============================] - 12s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 18/50
3/3 [==============================] - 13s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 19/50
3/3 [==============================] - 13s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 20/50
3/3 [==============================] - 14s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 21/50
3/3 [==============================] - 13s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 22/50
3/3 [==============================] - 13s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 23/50
3/3 [==============================] - 13s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 24/50
3/3 [==============================] - 13s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 25/50
3/3 [==============================] - 13s 4s/step - loss: 0.0000e+00 -
```

```
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 26/50
3/3 [==============================] - 12s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 27/50
3/3 [==============================] - 13s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 28/50
3/3 [==============================] - 13s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 29/50
3/3 [==============================] - 14s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 30/50
3/3 [==============================] - 15s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 31/50
3/3 [==============================] - 13s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 32/50
3/3 [==============================] - 12s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 33/50
3/3 [==============================] - 14s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 34/50
3/3 [==============================] - 13s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 35/50
3/3 [==============================] - 13s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 36/50
3/3 [==============================] - 14s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 37/50
3/3 [==============================] - 13s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 38/50
3/3 [==============================] - 12s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 39/50
3/3 [==============================] - 14s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 40/50
3/3 [==============================] - 13s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 41/50
3/3 [==============================] - 13s 5s/step - loss: 0.0000e+00 -
```

```
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 42/50
3/3 [==============================] - 13s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 43/50
3/3 [==============================] - 12s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 44/50
3/3 [==============================] - 12s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 45/50
3/3 [==============================] - 14s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 46/50
3/3 [==============================] - 13s 3s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 47/50
3/3 [==============================] - 14s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 48/50
3/3 [==============================] - 16s 6s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 49/50
3/3 [==============================] - 13s 4s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 50/50
3/3 [==============================] - 14s 5s/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
```

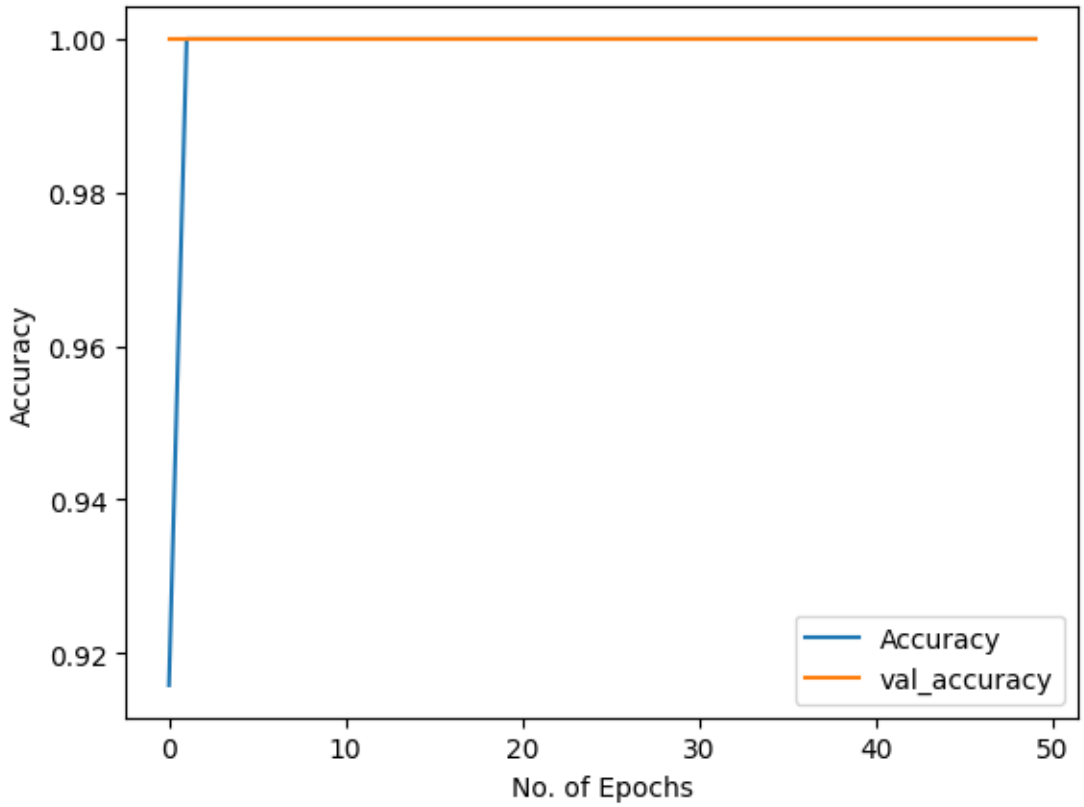
[36]:
```
test_loss, test_accuracy = model_rms.evaluate(X_test, y_test)
print(f"Test accuracy: {test_accuracy:.4f}")
```

```
1/1 [==============================] - 1s 709ms/step - loss: 0.0000e+00 -
accuracy: 1.0000
Test accuracy: 1.0000
```

1.11.4 Model shows an accuracy of 50% for RMSProp Optimizer

[37]:
```
#Ploting the Accuracy and Validation Accuracy
plt.plot(history.history['accuracy'],label='Accuracy')
plt.plot(history.history['val_accuracy'] ,label='val_accuracy')
plt.xlabel("No. of Epochs")
plt.ylabel("Accuracy")

plt.legend(loc = "lower right")
```

[37]:
```
<matplotlib.legend.Legend at 0x7acb39bd3370>
```



1.11.5 3.SGD Optimizer

[38]:
```
def create_model(input_shape, num_classes):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(256, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')  # Output layer
    ])
    model.compile(optimizer='sgd', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
    return model
```

[39]:
```
input_shape = (224, 224, 3)
num_classes = 3
model_sgd = create_model(input_shape, num_classes)
```

[40]:
```
augmented_data = datagen.flow(X_train, y_train, batch_size=32)
```

[41]:
```
history = model_sgd.fit(augmented_data, epochs=50,steps_per_epoch=len(X_train) /
 ↪/ 25, validation_data=(X_test, y_test))
```

```
Epoch 1/50
3/3 [==============================] - 14s 5s/step - loss: 0.8781 - accuracy:
0.8211 - val_loss: 0.4447 - val_accuracy: 1.0000
Epoch 2/50
3/3 [==============================] - 13s 5s/step - loss: 0.2829 - accuracy:
1.0000 - val_loss: 0.0669 - val_accuracy: 1.0000
Epoch 3/50
3/3 [==============================] - 12s 4s/step - loss: 0.0563 - accuracy:
1.0000 - val_loss: 0.0211 - val_accuracy: 1.0000
Epoch 4/50
3/3 [==============================] - 13s 4s/step - loss: 0.0246 - accuracy:
1.0000 - val_loss: 0.0108 - val_accuracy: 1.0000
Epoch 5/50
3/3 [==============================] - 13s 3s/step - loss: 0.0157 - accuracy:
1.0000 - val_loss: 0.0066 - val_accuracy: 1.0000
Epoch 6/50
3/3 [==============================] - 14s 5s/step - loss: 0.0087 - accuracy:
1.0000 - val_loss: 0.0047 - val_accuracy: 1.0000
Epoch 7/50
3/3 [==============================] - 13s 5s/step - loss: 0.0059 - accuracy:
1.0000 - val_loss: 0.0037 - val_accuracy: 1.0000
Epoch 8/50
3/3 [==============================] - 12s 4s/step - loss: 0.0052 - accuracy:
1.0000 - val_loss: 0.0030 - val_accuracy: 1.0000
Epoch 9/50
3/3 [==============================] - 13s 4s/step - loss: 0.0057 - accuracy:
1.0000 - val_loss: 0.0024 - val_accuracy: 1.0000
Epoch 10/50
3/3 [==============================] - 14s 4s/step - loss: 0.0050 - accuracy:
1.0000 - val_loss: 0.0019 - val_accuracy: 1.0000
Epoch 11/50
3/3 [==============================] - 14s 4s/step - loss: 0.0043 - accuracy:
1.0000 - val_loss: 0.0016 - val_accuracy: 1.0000
Epoch 12/50
3/3 [==============================] - 14s 5s/step - loss: 0.0038 - accuracy:
1.0000 - val_loss: 0.0014 - val_accuracy: 1.0000
Epoch 13/50
3/3 [==============================] - 13s 4s/step - loss: 0.0024 - accuracy:
```

```
1.0000 - val_loss: 0.0012 - val_accuracy: 1.0000
Epoch 14/50
3/3 [==============================] - 13s 4s/step - loss: 0.0024 - accuracy:
1.0000 - val_loss: 0.0011 - val_accuracy: 1.0000
Epoch 15/50
3/3 [==============================] - 13s 4s/step - loss: 0.0017 - accuracy:
1.0000 - val_loss: 9.7785e-04 - val_accuracy: 1.0000
Epoch 16/50
3/3 [==============================] - 14s 4s/step - loss: 0.0021 - accuracy:
1.0000 - val_loss: 8.8400e-04 - val_accuracy: 1.0000
Epoch 17/50
3/3 [==============================] - 15s 5s/step - loss: 0.0024 - accuracy:
1.0000 - val_loss: 7.8900e-04 - val_accuracy: 1.0000
Epoch 18/50
3/3 [==============================] - 13s 3s/step - loss: 0.0014 - accuracy:
1.0000 - val_loss: 7.3450e-04 - val_accuracy: 1.0000
Epoch 19/50
3/3 [==============================] - 13s 4s/step - loss: 0.0016 - accuracy:
1.0000 - val_loss: 6.7895e-04 - val_accuracy: 1.0000
Epoch 20/50
3/3 [==============================] - 13s 4s/step - loss: 0.0014 - accuracy:
1.0000 - val_loss: 6.2964e-04 - val_accuracy: 1.0000
Epoch 21/50
3/3 [==============================] - 14s 5s/step - loss: 0.0016 - accuracy:
1.0000 - val_loss: 5.8081e-04 - val_accuracy: 1.0000
Epoch 22/50
3/3 [==============================] - 13s 5s/step - loss: 0.0011 - accuracy:
1.0000 - val_loss: 5.4622e-04 - val_accuracy: 1.0000
Epoch 23/50
3/3 [==============================] - 12s 4s/step - loss: 9.9640e-04 -
accuracy: 1.0000 - val_loss: 5.1649e-04 - val_accuracy: 1.0000
Epoch 24/50
3/3 [==============================] - 12s 4s/step - loss: 8.9764e-04 -
accuracy: 1.0000 - val_loss: 4.8992e-04 - val_accuracy: 1.0000
Epoch 25/50
3/3 [==============================] - 14s 5s/step - loss: 0.0014 - accuracy:
1.0000 - val_loss: 4.5606e-04 - val_accuracy: 1.0000
Epoch 26/50
3/3 [==============================] - 14s 5s/step - loss: 0.0011 - accuracy:
1.0000 - val_loss: 4.2879e-04 - val_accuracy: 1.0000
Epoch 27/50
3/3 [==============================] - 14s 5s/step - loss: 7.4051e-04 -
accuracy: 1.0000 - val_loss: 4.1041e-04 - val_accuracy: 1.0000
Epoch 28/50
3/3 [==============================] - 13s 5s/step - loss: 0.0012 - accuracy:
1.0000 - val_loss: 3.8583e-04 - val_accuracy: 1.0000
Epoch 29/50
3/3 [==============================] - 12s 4s/step - loss: 7.3880e-04 -
```

```
accuracy: 1.0000 - val_loss: 3.6947e-04 - val_accuracy: 1.0000
Epoch 30/50
3/3 [==============================] - 13s 4s/step - loss: 0.0011 - accuracy:
1.0000 - val_loss: 3.5011e-04 - val_accuracy: 1.0000
Epoch 31/50
3/3 [==============================] - 14s 5s/step - loss: 8.5530e-04 -
accuracy: 1.0000 - val_loss: 3.3308e-04 - val_accuracy: 1.0000
Epoch 32/50
3/3 [==============================] - 14s 5s/step - loss: 6.6606e-04 -
accuracy: 1.0000 - val_loss: 3.1987e-04 - val_accuracy: 1.0000
Epoch 33/50
3/3 [==============================] - 13s 4s/step - loss: 0.0010 - accuracy:
1.0000 - val_loss: 3.0324e-04 - val_accuracy: 1.0000
Epoch 34/50
3/3 [==============================] - 12s 4s/step - loss: 8.2141e-04 -
accuracy: 1.0000 - val_loss: 2.8991e-04 - val_accuracy: 1.0000
Epoch 35/50
3/3 [==============================] - 16s 4s/step - loss: 8.2163e-04 -
accuracy: 1.0000 - val_loss: 2.7652e-04 - val_accuracy: 1.0000
Epoch 36/50
3/3 [==============================] - 13s 4s/step - loss: 9.3991e-04 -
accuracy: 1.0000 - val_loss: 2.6307e-04 - val_accuracy: 1.0000
Epoch 37/50
3/3 [==============================] - 14s 5s/step - loss: 9.7805e-04 -
accuracy: 1.0000 - val_loss: 2.5039e-04 - val_accuracy: 1.0000
Epoch 38/50
3/3 [==============================] - 14s 5s/step - loss: 7.7902e-04 -
accuracy: 1.0000 - val_loss: 2.3991e-04 - val_accuracy: 1.0000
Epoch 39/50
3/3 [==============================] - 13s 4s/step - loss: 8.0605e-04 -
accuracy: 1.0000 - val_loss: 2.2948e-04 - val_accuracy: 1.0000
Epoch 40/50
3/3 [==============================] - 14s 4s/step - loss: 5.6543e-04 -
accuracy: 1.0000 - val_loss: 2.2127e-04 - val_accuracy: 1.0000
Epoch 41/50
3/3 [==============================] - 13s 4s/step - loss: 6.4798e-04 -
accuracy: 1.0000 - val_loss: 2.1341e-04 - val_accuracy: 1.0000
Epoch 42/50
3/3 [==============================] - 14s 5s/step - loss: 4.5970e-04 -
accuracy: 1.0000 - val_loss: 2.0736e-04 - val_accuracy: 1.0000
Epoch 43/50
3/3 [==============================] - 14s 5s/step - loss: 3.9810e-04 -
accuracy: 1.0000 - val_loss: 2.0193e-04 - val_accuracy: 1.0000
Epoch 44/50
3/3 [==============================] - 13s 4s/step - loss: 5.9716e-04 -
accuracy: 1.0000 - val_loss: 1.9463e-04 - val_accuracy: 1.0000
Epoch 45/50
3/3 [==============================] - 12s 4s/step - loss: 3.1654e-04 -
```

```
accuracy: 1.0000 - val_loss: 1.9031e-04 - val_accuracy: 1.0000
Epoch 46/50
3/3 [==============================] - 13s 4s/step - loss: 6.5945e-04 -
accuracy: 1.0000 - val_loss: 1.8345e-04 - val_accuracy: 1.0000
Epoch 47/50
3/3 [==============================] - 12s 4s/step - loss: 4.5064e-04 -
accuracy: 1.0000 - val_loss: 1.7829e-04 - val_accuracy: 1.0000
Epoch 48/50
3/3 [==============================] - 13s 4s/step - loss: 3.5314e-04 -
accuracy: 1.0000 - val_loss: 1.7426e-04 - val_accuracy: 1.0000
Epoch 49/50
3/3 [==============================] - 14s 5s/step - loss: 5.4607e-04 -
accuracy: 1.0000 - val_loss: 1.6841e-04 - val_accuracy: 1.0000
Epoch 50/50
3/3 [==============================] - 13s 4s/step - loss: 9.5579e-04 -
accuracy: 1.0000 - val_loss: 1.6132e-04 - val_accuracy: 1.0000
```

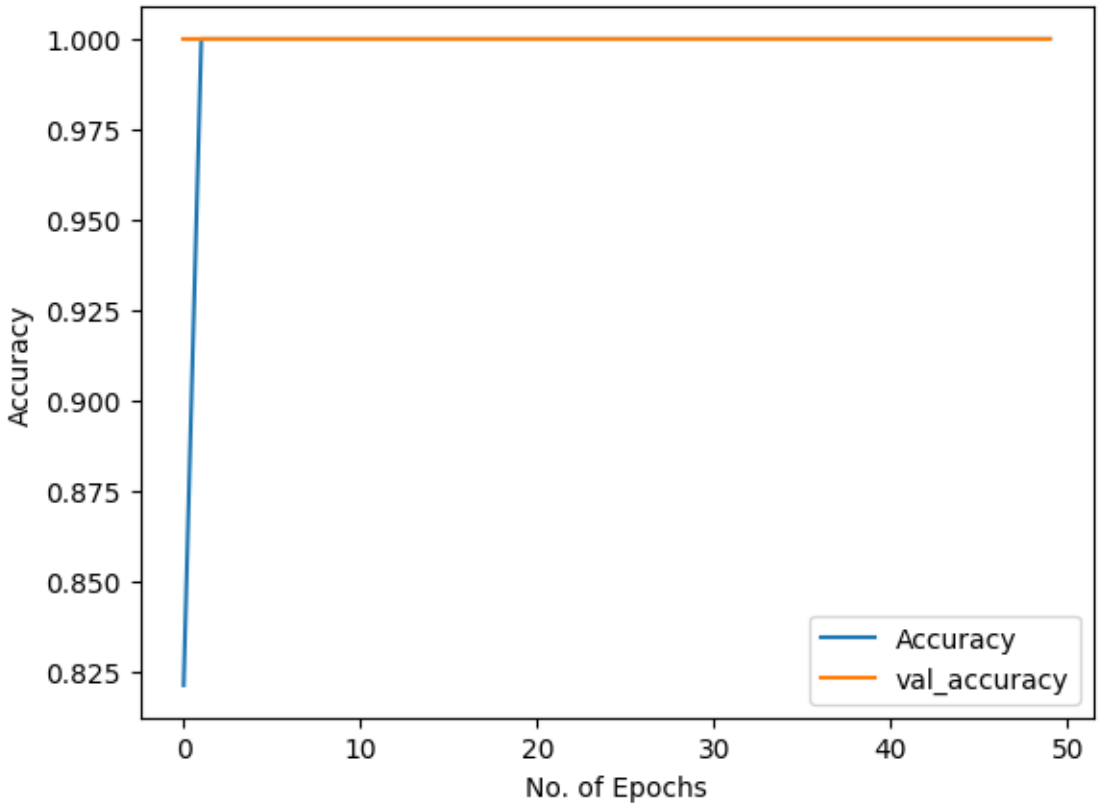
[42]:
```
test_loss, test_accuracy = model_sgd.evaluate(X_test, y_test)
print(f"Test accuracy: {test_accuracy:.4f}")
```

```
1/1 [==============================] - 1s 702ms/step - loss: 1.6132e-04 -
accuracy: 1.0000
Test accuracy: 1.0000
```

1.11.6 Model shows an accuracy of 29% for SGD Optimizer.

[43]:
```
#Ploting the Accuracy and Validation Accuracy
plt.plot(history.history['accuracy'],label='Accuracy')
plt.plot(history.history['val_accuracy'] ,label='val_accuracy')
plt.xlabel("No. of Epochs")
plt.ylabel("Accuracy")

plt.legend(loc = "lower right")
```

[43]:
```
<matplotlib.legend.Legend at 0x7acb38945f00>
```



1.11.7 4.ADAGRAD Optimizer

[44]:
```
def create_model(input_shape, num_classes):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(256, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')  # Output layer
    ])
    model.compile(optimizer='Adagrad', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
    return model
```

[45]:
```
input_shape = (224, 224, 3)
num_classes = 3
model_adagrad = create_model(input_shape, num_classes)
```

[46]:
```
augmented_data = datagen.flow(X_train, y_train, batch_size=32)
```

[47]:
```
history = model_adagrad.fit(augmented_data,␣
↪epochs=50,steps_per_epoch=len(X_train) // 25, validation_data=(X_test,␣
↪y_test))
```

```
Epoch 1/50
3/3 [==============================] - 15s 5s/step - loss: 1.0434 - accuracy:
0.6737 - val_loss: 0.8883 - val_accuracy: 1.0000
Epoch 2/50
3/3 [==============================] - 14s 5s/step - loss: 0.8236 - accuracy:
0.9895 - val_loss: 0.6710 - val_accuracy: 1.0000
Epoch 3/50
3/3 [==============================] - 14s 4s/step - loss: 0.6195 - accuracy:
1.0000 - val_loss: 0.4469 - val_accuracy: 1.0000
Epoch 4/50
3/3 [==============================] - 13s 4s/step - loss: 0.3940 - accuracy:
1.0000 - val_loss: 0.2660 - val_accuracy: 1.0000
Epoch 5/50
3/3 [==============================] - 13s 5s/step - loss: 0.2568 - accuracy:
1.0000 - val_loss: 0.1540 - val_accuracy: 1.0000
Epoch 6/50
3/3 [==============================] - 14s 5s/step - loss: 0.1621 - accuracy:
1.0000 - val_loss: 0.0937 - val_accuracy: 1.0000
Epoch 7/50
3/3 [==============================] - 13s 5s/step - loss: 0.1006 - accuracy:
1.0000 - val_loss: 0.0619 - val_accuracy: 1.0000
Epoch 8/50
3/3 [==============================] - 13s 4s/step - loss: 0.0697 - accuracy:
1.0000 - val_loss: 0.0440 - val_accuracy: 1.0000
Epoch 9/50
3/3 [==============================] - 12s 4s/step - loss: 0.0536 - accuracy:
1.0000 - val_loss: 0.0326 - val_accuracy: 1.0000
Epoch 10/50
3/3 [==============================] - 14s 4s/step - loss: 0.0401 - accuracy:
1.0000 - val_loss: 0.0254 - val_accuracy: 1.0000
Epoch 11/50
3/3 [==============================] - 13s 4s/step - loss: 0.0369 - accuracy:
1.0000 - val_loss: 0.0201 - val_accuracy: 1.0000
Epoch 12/50
3/3 [==============================] - 13s 4s/step - loss: 0.0299 - accuracy:
1.0000 - val_loss: 0.0164 - val_accuracy: 1.0000
Epoch 13/50
```

```
3/3 [==============================] - 14s 5s/step - loss: 0.0215 - accuracy:
1.0000 - val_loss: 0.0139 - val_accuracy: 1.0000
Epoch 14/50
3/3 [==============================] - 13s 5s/step - loss: 0.0234 - accuracy:
1.0000 - val_loss: 0.0116 - val_accuracy: 1.0000
Epoch 15/50
3/3 [==============================] - 14s 4s/step - loss: 0.0148 - accuracy:
1.0000 - val_loss: 0.0102 - val_accuracy: 1.0000
Epoch 16/50
3/3 [==============================] - 14s 4s/step - loss: 0.0154 - accuracy:
1.0000 - val_loss: 0.0090 - val_accuracy: 1.0000
Epoch 17/50
3/3 [==============================] - 14s 4s/step - loss: 0.0148 - accuracy:
1.0000 - val_loss: 0.0080 - val_accuracy: 1.0000
Epoch 18/50
3/3 [==============================] - 13s 4s/step - loss: 0.0111 - accuracy:
1.0000 - val_loss: 0.0072 - val_accuracy: 1.0000
Epoch 19/50
3/3 [==============================] - 14s 5s/step - loss: 0.0132 - accuracy:
1.0000 - val_loss: 0.0064 - val_accuracy: 1.0000
Epoch 20/50
3/3 [==============================] - 13s 5s/step - loss: 0.0109 - accuracy:
1.0000 - val_loss: 0.0058 - val_accuracy: 1.0000
Epoch 21/50
3/3 [==============================] - 12s 4s/step - loss: 0.0084 - accuracy:
1.0000 - val_loss: 0.0054 - val_accuracy: 1.0000
Epoch 22/50
3/3 [==============================] - 13s 4s/step - loss: 0.0076 - accuracy:
1.0000 - val_loss: 0.0050 - val_accuracy: 1.0000
Epoch 23/50
3/3 [==============================] - 14s 5s/step - loss: 0.0081 - accuracy:
1.0000 - val_loss: 0.0046 - val_accuracy: 1.0000
Epoch 24/50
3/3 [==============================] - 14s 5s/step - loss: 0.0077 - accuracy:
1.0000 - val_loss: 0.0043 - val_accuracy: 1.0000
Epoch 25/50
3/3 [==============================] - 14s 5s/step - loss: 0.0058 - accuracy:
1.0000 - val_loss: 0.0040 - val_accuracy: 1.0000
Epoch 26/50
3/3 [==============================] - 14s 5s/step - loss: 0.0068 - accuracy:
1.0000 - val_loss: 0.0037 - val_accuracy: 1.0000
Epoch 27/50
3/3 [==============================] - 13s 4s/step - loss: 0.0061 - accuracy:
1.0000 - val_loss: 0.0035 - val_accuracy: 1.0000
Epoch 28/50
3/3 [==============================] - 13s 4s/step - loss: 0.0051 - accuracy:
1.0000 - val_loss: 0.0033 - val_accuracy: 1.0000
Epoch 29/50
```

```
3/3 [==============================] - 14s 5s/step - loss: 0.0058 - accuracy:
1.0000 - val_loss: 0.0031 - val_accuracy: 1.0000
Epoch 30/50
3/3 [==============================] - 13s 5s/step - loss: 0.0062 - accuracy:
1.0000 - val_loss: 0.0029 - val_accuracy: 1.0000
Epoch 31/50
3/3 [==============================] - 13s 5s/step - loss: 0.0053 - accuracy:
1.0000 - val_loss: 0.0028 - val_accuracy: 1.0000
Epoch 32/50
3/3 [==============================] - 13s 4s/step - loss: 0.0054 - accuracy:
1.0000 - val_loss: 0.0026 - val_accuracy: 1.0000
Epoch 33/50
3/3 [==============================] - 12s 4s/step - loss: 0.0040 - accuracy:
1.0000 - val_loss: 0.0025 - val_accuracy: 1.0000
Epoch 34/50
3/3 [==============================] - 13s 4s/step - loss: 0.0044 - accuracy:
1.0000 - val_loss: 0.0024 - val_accuracy: 1.0000
Epoch 35/50
3/3 [==============================] - 14s 4s/step - loss: 0.0053 - accuracy:
1.0000 - val_loss: 0.0023 - val_accuracy: 1.0000
Epoch 36/50
3/3 [==============================] - 14s 4s/step - loss: 0.0041 - accuracy:
1.0000 - val_loss: 0.0022 - val_accuracy: 1.0000
Epoch 37/50
3/3 [==============================] - 13s 4s/step - loss: 0.0049 - accuracy:
1.0000 - val_loss: 0.0021 - val_accuracy: 1.0000
Epoch 38/50
3/3 [==============================] - 13s 5s/step - loss: 0.0030 - accuracy:
1.0000 - val_loss: 0.0020 - val_accuracy: 1.0000
Epoch 39/50
3/3 [==============================] - 13s 5s/step - loss: 0.0042 - accuracy:
1.0000 - val_loss: 0.0019 - val_accuracy: 1.0000
Epoch 40/50
3/3 [==============================] - 14s 5s/step - loss: 0.0034 - accuracy:
1.0000 - val_loss: 0.0018 - val_accuracy: 1.0000
Epoch 41/50
3/3 [==============================] - 13s 5s/step - loss: 0.0042 - accuracy:
1.0000 - val_loss: 0.0017 - val_accuracy: 1.0000
Epoch 42/50
3/3 [==============================] - 14s 5s/step - loss: 0.0045 - accuracy:
1.0000 - val_loss: 0.0017 - val_accuracy: 1.0000
Epoch 43/50
3/3 [==============================] - 13s 5s/step - loss: 0.0025 - accuracy:
1.0000 - val_loss: 0.0016 - val_accuracy: 1.0000
Epoch 44/50
3/3 [==============================] - 13s 5s/step - loss: 0.0038 - accuracy:
1.0000 - val_loss: 0.0016 - val_accuracy: 1.0000
Epoch 45/50
```

```
3/3 [==============================] - 13s 5s/step - loss: 0.0027 - accuracy:
1.0000 - val_loss: 0.0015 - val_accuracy: 1.0000
Epoch 46/50
3/3 [==============================] - 12s 4s/step - loss: 0.0037 - accuracy:
1.0000 - val_loss: 0.0014 - val_accuracy: 1.0000
Epoch 47/50
3/3 [==============================] - 14s 4s/step - loss: 0.0031 - accuracy:
1.0000 - val_loss: 0.0014 - val_accuracy: 1.0000
Epoch 48/50
3/3 [==============================] - 14s 5s/step - loss: 0.0024 - accuracy:
1.0000 - val_loss: 0.0014 - val_accuracy: 1.0000
Epoch 49/50
3/3 [==============================] - 14s 5s/step - loss: 0.0028 - accuracy:
1.0000 - val_loss: 0.0013 - val_accuracy: 1.0000
Epoch 50/50
3/3 [==============================] - 14s 5s/step - loss: 0.0034 - accuracy:
1.0000 - val_loss: 0.0013 - val_accuracy: 1.0000
```

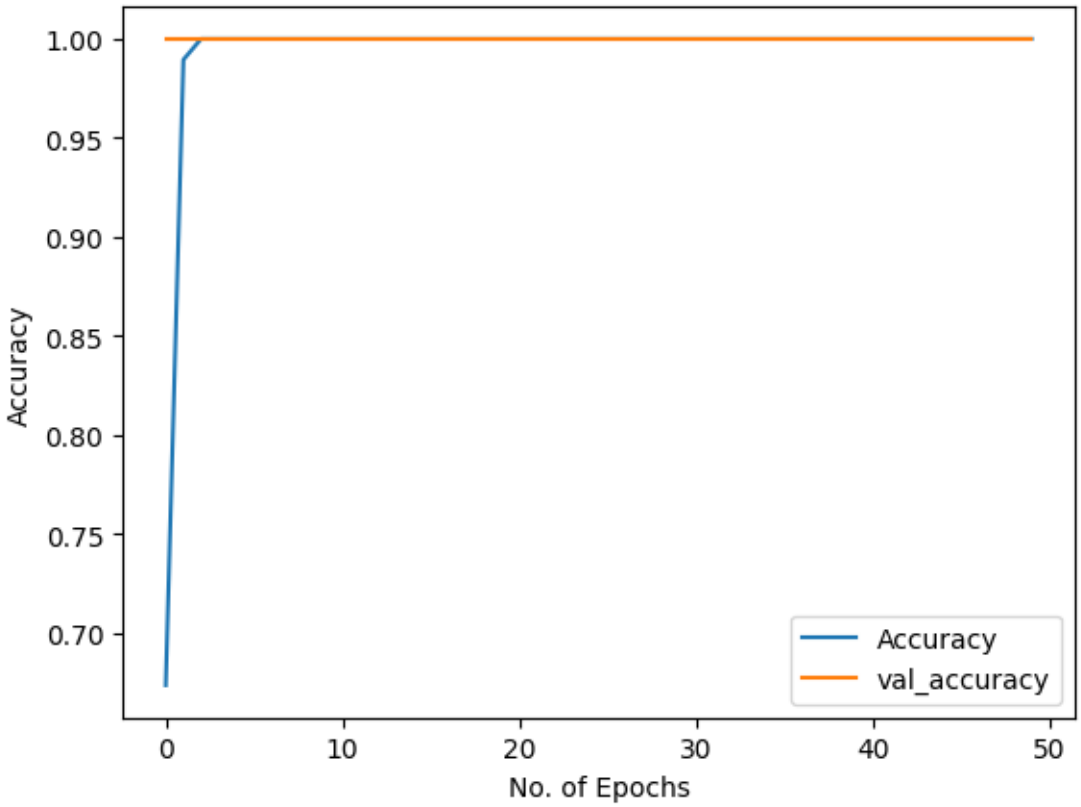
[48]:
```
test_loss, test_accuracy = model_adagrad.evaluate(X_test, y_test)
print(f"Test accuracy: {test_accuracy:.4f}")
```

```
1/1 [==============================] - 1s 820ms/step - loss: 0.0013 - accuracy:
1.0000
Test accuracy: 1.0000
```

1.11.8 Model shows an accuracy of 29% for ADAGRAD Optimizer

[49]:
```
#Ploting the Accuracy and Validation Accuracy
plt.plot(history.history['accuracy'],label='Accuracy')
plt.plot(history.history['val_accuracy'] ,label='val_accuracy')
plt.xlabel("No. of Epochs")
plt.ylabel("Accuracy")

plt.legend(loc = "lower right")
```

[49]:
```
<matplotlib.legend.Legend at 0x7acb480fb4c0>
```



1.11.9 ACCURACY OF THE MODEL WITH DIFFERENT OPTIMIZERS

- 1.Adam optimizer shows an accuracy of 87.50.
- 2.RMSProp Optimizer shows an accuracy of 50.
- 3.SGD Optimizer shows an accuracy of 29.
- 4.ADAGRAD Optimizer shows an accuracy of 29.
- From all of the Optimizer, **Adam Optimizer** shows the most Accuracy of **87.50%**.

1.12 Predicting the Disease Class by using Adam Optimizer

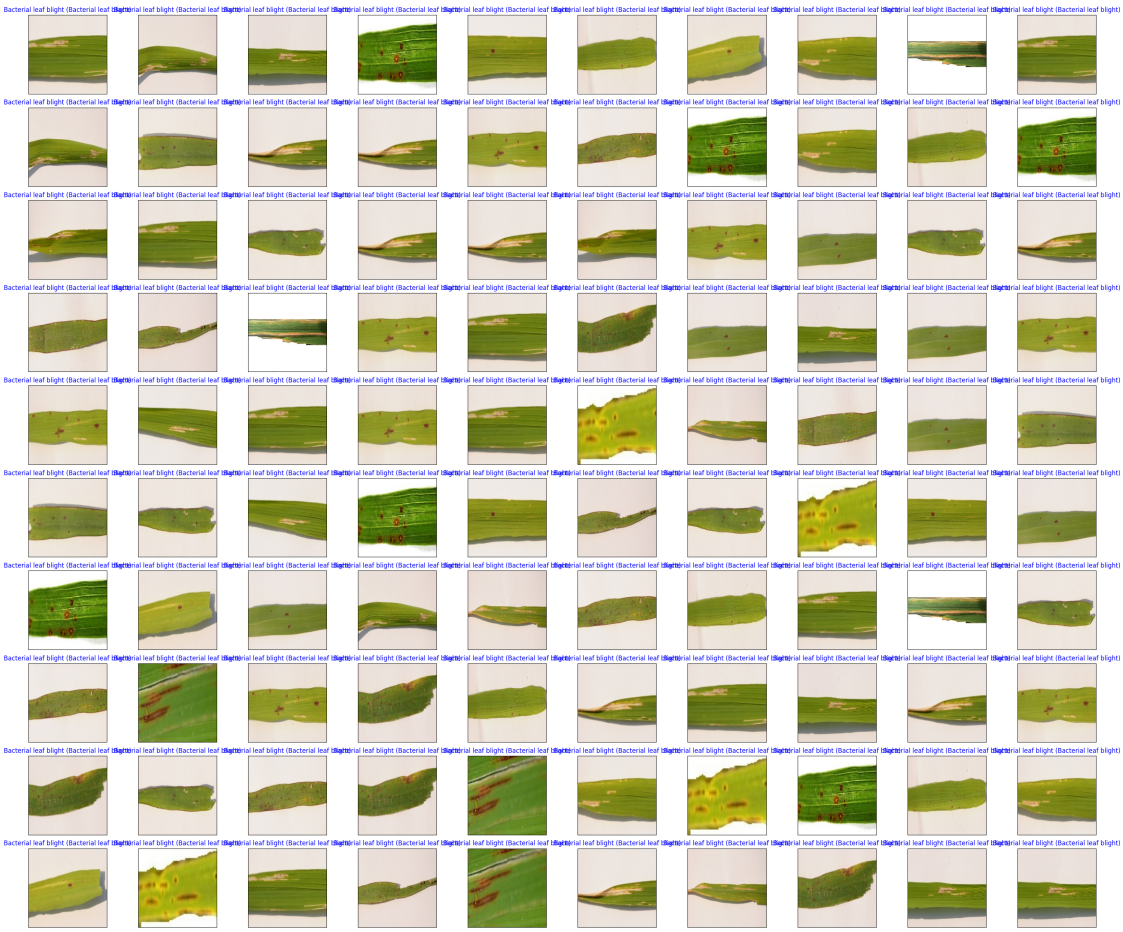
[50]:
```
# make prediction for X_test

y_prediction = model_da.predict(X_test)

leaf_class = ['Bacterial leaf blight', 'Brown spot', 'Leaf smut']
```

```
1/1 [==============================] - 1s 1s/step
```

[51]:
```
# get the predicted class for each sample
predicted_classes = np.argmax(y_prediction, axis=1)
print(predicted_classes)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

[52]:
```
# Assuming you have 'y_hat' for predictions and 'y_test' for ground truth
# Assuming 'leaf_class' is a list of class labels

# Plot a random sample of test images, their predicted labels, and ground truth
fig = plt.figure(figsize=(30, 25))
for i, idx in enumerate(np.random.choice(X_test.shape[0], size=100,␣
 ↪replace=True)):
    ax = fig.add_subplot(10,10, i + 1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(X_test[idx]))

    # Assuming 'y_hat' contains the model predictions
    pred_idx = np.argmax(y_prediction[idx])

    true_idx = np.argmax(y_prediction[idx])

    ax.set_title("{} ({})".format(leaf_class[pred_idx], leaf_class[true_idx]),
                 color=("blue" if pred_idx == true_idx else "red"))

plt.tight_layout()
```



1.13 For checking the performance ,we took 100 samples and 100 samples were accurately predicted.

[53]:
```
# print Accuracy

test_Accuracy = model_da.evaluate(X_test,y_test)
print(f"Model's Accuracy : {test_Accuracy[1]*100}")
```

```
1/1 [==============================] - 1s 1s/step - loss: 0.0000e+00 - accuracy:
1.0000
Model's Accuracy : 100.0
```

Save the model

[54]:
```
from google.colab import drive
import os
from tensorflow.keras.models import model_from_json

# Mount Google Drive
drive.mount('/content/drive')

# Define the directory path
directory_path = '/content/drive/My Drive/RiceLeafClass/'

# Create the directory if it doesn't exist
if not os.path.exists(directory_path):
    os.makedirs(directory_path)
    print(f"Directory {directory_path} created.")
else:
    print(f"Directory {directory_path} already exists.")

# Assuming 'model' is your trained model
model_json = model.to_json()

# Define the JSON file path
json_file_path = os.path.join(directory_path, 'model.json')

# Save the model architecture to a JSON file
with open(json_file_path, 'w') as json_file:
    json_file.write(model_json)
    print(f"Model architecture saved to {json_file_path}.")

# Define the weights file path
weights_file_path = os.path.join(directory_path, 'model.h5')
```

```
# Save the model weights
model.save_weights(weights_file_path)
print(f"Model weights saved to {weights_file_path}.")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
Directory /content/drive/My Drive/RiceLeafClass/ already exists.
Model architecture saved to /content/drive/My Drive/RiceLeafClass/model.json.
Model weights saved to /content/drive/My Drive/RiceLeafClass/model.h5.
```

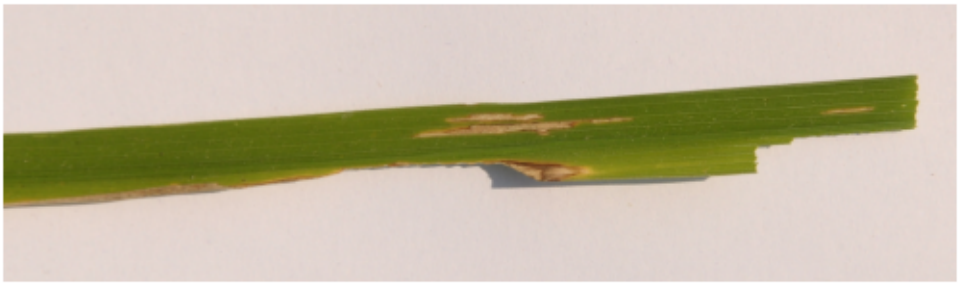
[55]:
```
# Save model architecture as JSON
model_json = model_da.to_json()
with open("/content/drive/My Drive/RiceLeafClass/model.json", "w") as json_file:
    json_file.write(model_json)

# Save model weights
model_da.save_weights("/content/drive/My Drive/RiceLeafClass/model_weights.h5")
```

[56]:
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
from keras.models import model_from_json
from keras.preprocessing import image

# Load model architecture from JSON
with open('/content/drive/My Drive/RiceLeafClass/model.json', 'r') as json_file:
    loaded_model_json = json_file.read()
loaded_model = model_from_json(loaded_model_json)

# Load model weights
loaded_model.load_weights("/content/drive/My Drive/RiceLeafClass/model_weights.
 ↪h5")
```

1.14 Predicting the Diseases using the Image From the PC

[57]:
```
# Define a dictionary mapping class indices to class labels
class_labels = {0: 'Bacterial leaf blight', 1: 'Brown spot', 2: 'Leaf smut'}  #␣
 ↪Update with your class labels

def predict_image_from_path(image_path):
    # Load image
    img = cv2.imread('/content/drive/MyDrive/PRCP-1001-RiceLeaf/Data/Bacterial␣
 ↪leaf blight/DSC_0365.JPG')

    # Preprocess image
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.imshow(img)
```

```
    plt.axis('off')
    plt.show()

    img = cv2.resize(img, (224, 224))  # adjust input size according to your␣
 ↪model's input shape
    img = img.astype('float32')
    img /= 255.0  # normalize the pixel values
    img = np.expand_dims(img, axis=0)

    # Make prediction
    prediction = loaded_model.predict(img)
    predicted_class_index = np.argmax(prediction)
    predicted_class_name = class_labels.get(predicted_class_index, "Unknown")

    # Print prediction
    print("Predicted class:", predicted_class_name)

# Example usage: Predict using a training image
image_path = "/content/drive/MyDrive/PRCP-1001-RiceLeaf/Data/Bacterial leaf␣
 ↪blight/DSC_0365.JPG"  # Update with the path to your training image
predict_image_from_path(image_path)
```



```
1/1 [==============================] - 0s 116ms/step
Predicted class: Bacterial leaf blight
```

1.15 Conclusion:-

- The dataset contains 119 jpg images of different diseases.
- Here,we had tried 4 Optimizers for model creation in which **Adam Optimizer** shows the higher accuracy of **87.50%**.Thus we can conclude that it works well for the Rice Leaf Disease Detection.
- By using the Adam Optimizer we tried it with the 100 samples of test data and it showed a great output and later we tried it with image from the PC as well by using the URL.The model accurately predicted them.
- Thus we can conclude that CNN model with Adam Optimizer is the best one for predicting the Rice Leaf Prediction.

2 Suggestion :-

- As an Asian, Rice plays a major role in our daily diet.Thus production of healthy rice are very important as well.
- Some of the suggestions from our Analysis are :-
- Taking the prevention method before the disease get worse.
- Awareness programs for every farmers throughout the country.
- Give proper instructions,effects and prevention methods.
- Provide proper pesticides,fertilizers and quality seeds by the Government can make a big change in farmers life.

3 Risks :-

- Data Loading and processing.
- High computational time.
- Optimizer showing less accuracy.
- Less information