RICE LEAF DISEASE PREDICTION PROJECT USING CNN AND DATA AUGUMENTATION

```
# parimal
```

Agriculture is the main source of food in India. During the crop cycle, many plant diseases can occur and can affect the crop. Identification and classification of plant disease at the leaf level is a crucial part of crop management and is important in the early detection and diagnosis of plant diseases. Traditionally, the classification of plant diseases has been performed by farmers using manual methods like manual inspection or field observations. However, this process is time-consuming and error-prone, which is why there is a need for an intelligent system that can automatically classify images based on plant leaf diseases.

Bacterial leaf blight --> 40 images Brown spot --> 40 images Leaf smut --> 39 images

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

import seaborn as sns
```

```
from google.colab import drive
drive.mount('/content/drive')
# Need to be connect to google drive for loading dataset
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# importing most important deep learing libraries
from tensorflow.keras.models import Sequential # most imporant DL library
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing.image import  array_to_img, img_to_array, load_img
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense

import random
from sklearn import preprocessing
import tensorflow.keras as keras
from keras import regularizers

import tensorflow as tf
from tensorflow.keras.utils import to_categorical

import glob
import cv2
import os
import pathlib
```

```
# Set random seed for reproducibility
tf.random.set_seed(42)
```

```
data_dir ="/content/drive/MyDrive/Data"
data_dir = pathlib.Path(data_dir)

CLASS_NAMES = np.array(['Leaf Blight','Brown Spot','Leaf Smut'])

print('Class Names: ', CLASS_NAMES)
```

```
Class Names:  ['Leaf Blight' 'Brown Spot' 'Leaf Smut']
```

```
# Seperate train and test path
train_path = '/content/drive/MyDrive/Data'
test_path = '/content/drive/MyDrive/Data'
```

2. Data Augumentation:

```
# Define the parameters
BATCH_SIZE = 16
IMG_HEIGHT = 224
IMG_WIDTH = 224
```

```
# Data augmentation for training set
image_train_gen = ImageDataGenerator(rescale=1./255,
                                     zoom_range=0.50,
                                     rotation_range=45,
                                     horizontal_flip=True,
                                     width_shift_range=0.15,
                                     height_shift_range=0.15)
```

```
# Load and augment the training dataset
train_data_gen = image_train_gen.flow_from_directory(train_path,
                                                     shuffle=True,
                                                     batch_size=BATCH_SIZE,
                                                     target_size=(IMG_HEIGHT,IMG_WIDTH),
                                                     class_mode='categorical')
```

```
Found 119 images belonging to 3 classes.
```

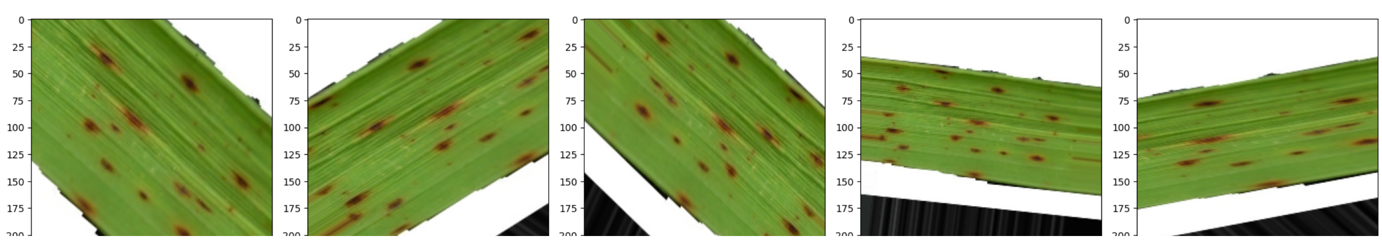
The ImageDataGenerator class has three methods flow(), flow_from_directory() and flow_from_dataframe() to read the images from a big numpy array and folders containing images. Here we will use flow_from_directory() as we have multiple images in our training set from each sub directory/class.

```
# Data normalization for test set (no augmentation)
img_val_gen = ImageDataGenerator(rescale=1./255) # test data
val_data_gen = img_val_gen.flow_from_directory(test_path,
                                               batch_size=BATCH_SIZE,
                                               target_size=(IMG_HEIGHT,IMG_WIDTH),
                                               class_mode='categorical')
```

```
Found 119 images belonging to 3 classes.
```

```
def plotImages(image_arr):
    fig,axes = plt.subplots(1, 5, figsize=(20,20))
    axes = axes.flatten()
    for img,ax in zip(image_arr,axes):
        ax.imshow(img)
    plt.tight_layout()
    plt.show()
```

```
# Plot a few training images-
img_array = [train_data_gen[0][0][0] for i in range(5)]
plotImages(img_array)
```



```
# plot a few val/testing  images
img_array  = [val_data_gen[0][0][0] for i in range(5)]
plotImages(img_array)
```

3.Model building


```
# Model building
#Instatiating A convnet: CNN

model = Sequential()
model.add(Conv2D(16, (3,3), input_shape=(224,224,3), activation="relu"))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(32, (3,3), activation="relu"))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(64, (3,3), activation="relu"))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(128,activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(3, activation="softmax"))


model.compile(
    optimizer = "adam",
    loss = "categorical_crossentropy",
    metrics = ['accuracy']
)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-67-2125fd759c33> in <cell line: 1>()
----> 1 model.compile(
      2     optimizer = "adam(lr= 0.001)",
      3     loss = "categorical_crossentropy",
      4     metrics = ['accuracy']
      5 )
```

1 frames

```
/usr/local/lib/python3.10/dist-packages/keras/saving/legacy/serialization.py in class_and_config_for_serialized_keras_object(config,
module_objects, custom_objects, printable_module_name)
    366     )
    367     if cls is None:
--> 368         raise ValueError(
    369             f"Unknown {printable_module_name}: '{class_name}'. "
    370             "Please ensure you are using a `keras.utils.custom_object_scope` "

ValueError: Unknown optimizer: 'adam(lr= 0.001)'. Please ensure you are using a `keras.utils.custom_object_scope` and that this object
is included in the scope. See https://www.tensorflow.org/guide/keras/save_and_serialize#registering_the_custom_object for details.
```

SEARCH STACK OVERFLOW

```
from keras.optimizers import Adam


model.compile(
    optimizer = Adam(lr=0.0001),
    loss='categorical_crossentropy', metrics=['accuracy'])
```

```
/usr/local/lib/python3.10/dist-packages/keras/optimizers/legacy/adam.py:117: UserWarning: The `lr` argument is deprecated, use `learnin
  super().__init__(name, **kwargs)
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 16)      448

 max_pooling2d (MaxPooling2D  (None, 111, 111, 16)     0
```

```
 )

 conv2d_1 (Conv2D)           (None, 109, 109, 32)      4640

 max_pooling2d_1 (MaxPooling  (None, 54, 54, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 52, 52, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 26, 26, 64)       0
 2D)

 flatten (Flatten)           (None, 43264)             0

 dropout (Dropout)           (None, 43264)             0

 dense (Dense)               (None, 128)               5537920

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 3)                 387

=================================================================
Total params: 5,561,891
Trainable params: 5,561,891
Non-trainable params: 0
_________________________________________________________________
```

```
# Total params: 5,561,891 TRAING PARAMETERS ARE GENERATED
```

```
EPOCHS=3
history = model.fit_generator(train_data_gen, epochs=EPOCHS, validation_data=val_data_gen)
```

```
<ipython-input-24-511f68dc7f00>:2: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use
  history = model.fit_generator(train_data_gen, epochs=EPOCHS, validation_data=val_data_gen)
Epoch 1/2
8/8 [==============================] - 69s 8s/step - loss: 2.0008 - accuracy: 0.2941 - val_loss: 1.1044 - val_accuracy: 0.3277
Epoch 2/2
8/8 [==============================] - 4s 543ms/step - loss: 1.1485 - accuracy: 0.3193 - val_loss: 1.0909 - val_accuracy: 0.4958
```

```
# ACCURACY is less apply more Epochs
```

```
EPOCHS=15
history = model.fit_generator(train_data_gen, epochs=EPOCHS, validation_data=val_data_gen)
```

```
<ipython-input-43-e3b79ad2518e>:2: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use
  history = model.fit_generator(train_data_gen, epochs=EPOCHS, validation_data=val_data_gen)
Epoch 1/15
8/8 [==============================] - 5s 623ms/step - loss: 0.3628 - accuracy: 0.8487 - val_loss: 0.1882 - val_accuracy: 0.9328
Epoch 2/15
8/8 [==============================] - 4s 560ms/step - loss: 0.4032 - accuracy: 0.8235 - val_loss: 0.2505 - val_accuracy: 0.9244
Epoch 3/15
8/8 [==============================] - 6s 741ms/step - loss: 0.3384 - accuracy: 0.8655 - val_loss: 0.1836 - val_accuracy: 0.9328
Epoch 4/15
8/8 [==============================] - 4s 539ms/step - loss: 0.3339 - accuracy: 0.8739 - val_loss: 0.1839 - val_accuracy: 0.9328
Epoch 5/15
8/8 [==============================] - 4s 552ms/step - loss: 0.3056 - accuracy: 0.8908 - val_loss: 0.2363 - val_accuracy: 0.9076
Epoch 6/15
8/8 [==============================] - 6s 740ms/step - loss: 0.3571 - accuracy: 0.8403 - val_loss: 0.1755 - val_accuracy: 0.9496
Epoch 7/15
8/8 [==============================] - 5s 650ms/step - loss: 0.3204 - accuracy: 0.8739 - val_loss: 0.2304 - val_accuracy: 0.9496
Epoch 8/15
8/8 [==============================] - 4s 530ms/step - loss: 0.3713 - accuracy: 0.8487 - val_loss: 0.1796 - val_accuracy: 0.9412
Epoch 9/15
8/8 [==============================] - 5s 600ms/step - loss: 0.3587 - accuracy: 0.8571 - val_loss: 0.2277 - val_accuracy: 0.9244
Epoch 10/15
8/8 [==============================] - 4s 534ms/step - loss: 0.3849 - accuracy: 0.8151 - val_loss: 0.1994 - val_accuracy: 0.8992
Epoch 11/15
8/8 [==============================] - 4s 577ms/step - loss: 0.3720 - accuracy: 0.8655 - val_loss: 0.2577 - val_accuracy: 0.8908
Epoch 12/15
8/8 [==============================] - 5s 577ms/step - loss: 0.4242 - accuracy: 0.8403 - val_loss: 0.2286 - val_accuracy: 0.9496
Epoch 13/15
8/8 [==============================] - 4s 559ms/step - loss: 0.3580 - accuracy: 0.8739 - val_loss: 0.2178 - val_accuracy: 0.8992
Epoch 14/15
8/8 [==============================] - 5s 724ms/step - loss: 0.3829 - accuracy: 0.8487 - val_loss: 0.1867 - val_accuracy: 0.9580
Epoch 15/15
8/8 [==============================] - 4s 559ms/step - loss: 0.3304 - accuracy: 0.8739 - val_loss: 0.1898 - val_accuracy: 0.9076
```

**After checkin epoch 67 the accuracy get decreases upto 0.71 so choose epochs 67 ** loss: 0.4924 - accuracy: 0.8487 - val_loss: 0.3235 - val_accuracy: 0.8992 \n actully afte 3 epoch we will get better acccuracy for test and traing dataset

4.chek Evaluate the model*

```
eval_loss, eval_accuracy = model.evaluate(val_data_gen)
print(f"Evaluation loss: {eval_loss:.4f}")
print(f"Evaluation accuracy: {eval_accuracy:.4f}")
```

```
8/8 [==============================] - 2s 181ms/step - loss: 0.1898 - accuracy: 0.9076
Evaluation loss: 0.1898
Evaluation accuracy: 0.9076
```

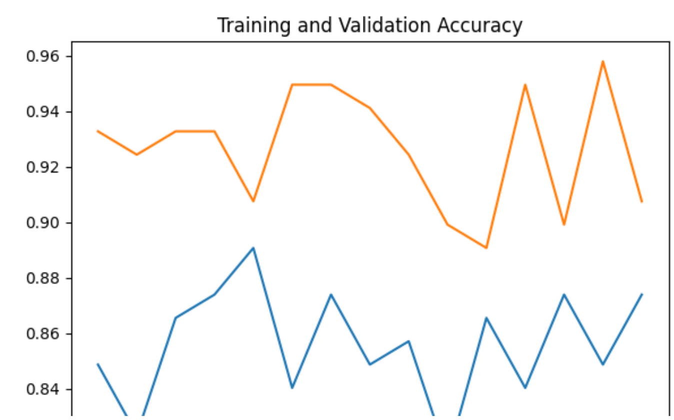
```
# Plot training and validation graphs
acc = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

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


print("test/val accuracy:", val_accuracy )
print("traing accuracy:",acc)
```

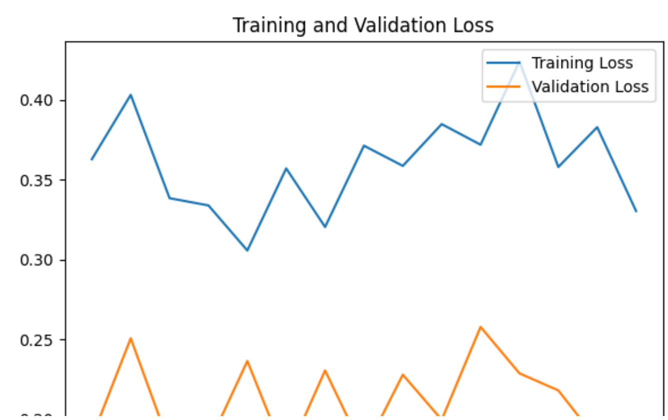
```
test/val accuracy: [0.9327731132507324, 0.924369752407074, 0.9327731132507324, 0.9327731132507324, 0.9075630307197571, 0.94957983493804
traing accuracy: [0.848739504814148, 0.8235294222831726, 0.8655462265014648, 0.8739495873451233, 0.8907563090324402, 0.8403361439704895
```

```
epochs_range = range(EPOCHS)

epochs_range
```

```
range(0, 15)
```

```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.plot(epochs_range,acc,label='Training Accuracy')
plt.plot(epochs_range,val_accuracy,label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1,2,2)
plt.plot(epochs_range,loss,label='Training Loss')
plt.plot(epochs_range,val_loss,label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```





if we use more epochs more than 50 loss get reduced and accuracy also get increases

```
EPOCHS=71
history = model.fit_generator(train_data_gen, epochs=EPOCHS, validation_data=val_data_gen)
```

```
<ipython-input-49-685cb9c937d1>:2: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please u
  history = model.fit_generator(train_data_gen, epochs=EPOCHS, validation_data=val_data_gen)
Epoch 1/71
8/8 [==============================] - 5s 655ms/step - loss: 0.2716 - accuracy: 0.8992 - val_loss: 0.1729 - val_accuracy: 0.9664
Epoch 2/71
8/8 [==============================] - 5s 705ms/step - loss: 0.2982 - accuracy: 0.8824 - val_loss: 0.2168 - val_accuracy: 0.8992
Epoch 3/71
8/8 [==============================] - 4s 536ms/step - loss: 0.3159 - accuracy: 0.8571 - val_loss: 0.1341 - val_accuracy: 0.9748
Epoch 4/71
8/8 [==============================] - 6s 778ms/step - loss: 0.2548 - accuracy: 0.8908 - val_loss: 0.3249 - val_accuracy: 0.8908
Epoch 5/71
8/8 [==============================] - 4s 562ms/step - loss: 0.3510 - accuracy: 0.8655 - val_loss: 0.2005 - val_accuracy: 0.9244
Epoch 6/71
8/8 [==============================] - 5s 631ms/step - loss: 0.3282 - accuracy: 0.9244 - val_loss: 0.1512 - val_accuracy: 0.9580
Epoch 7/71
8/8 [==============================] - 5s 670ms/step - loss: 0.3832 - accuracy: 0.8487 - val_loss: 0.1322 - val_accuracy: 0.9664
Epoch 8/71
8/8 [==============================] - 4s 537ms/step - loss: 0.3773 - accuracy: 0.8571 - val_loss: 0.2173 - val_accuracy: 0.9412
Epoch 9/71
8/8 [==============================] - 5s 718ms/step - loss: 0.3607 - accuracy: 0.8487 - val_loss: 0.1797 - val_accuracy: 0.9076
Epoch 10/71
8/8 [==============================] - 4s 562ms/step - loss: 0.4176 - accuracy: 0.8319 - val_loss: 0.1814 - val_accuracy: 0.9580
Epoch 11/71
8/8 [==============================] - 5s 705ms/step - loss: 0.2958 - accuracy: 0.8739 - val_loss: 0.2202 - val_accuracy: 0.9412
Epoch 12/71
8/8 [==============================] - 9s 1s/step - loss: 0.2209 - accuracy: 0.9412 - val_loss: 0.1582 - val_accuracy: 0.9664
Epoch 13/71
8/8 [==============================] - 4s 532ms/step - loss: 0.3192 - accuracy: 0.8992 - val_loss: 0.1624 - val_accuracy: 0.9412
Epoch 14/71
8/8 [==============================] - 4s 545ms/step - loss: 0.4117 - accuracy: 0.8403 - val_loss: 0.3841 - val_accuracy: 0.8151
Epoch 15/71
8/8 [==============================] - 6s 724ms/step - loss: 0.4570 - accuracy: 0.8235 - val_loss: 0.1469 - val_accuracy: 0.9580
Epoch 16/71
8/8 [==============================] - 4s 539ms/step - loss: 0.2837 - accuracy: 0.9244 - val_loss: 0.2144 - val_accuracy: 0.9076
Epoch 17/71
8/8 [==============================] - 5s 713ms/step - loss: 0.3002 - accuracy: 0.8571 - val_loss: 0.1555 - val_accuracy: 0.9496
Epoch 18/71
8/8 [==============================] - 4s 567ms/step - loss: 0.2381 - accuracy: 0.8908 - val_loss: 0.1177 - val_accuracy: 0.9748
Epoch 19/71
8/8 [==============================] - 5s 646ms/step - loss: 0.3062 - accuracy: 0.8655 - val_loss: 0.2265 - val_accuracy: 0.8992
Epoch 20/71
8/8 [==============================] - 4s 534ms/step - loss: 0.2953 - accuracy: 0.8739 - val_loss: 0.1411 - val_accuracy: 0.9496
Epoch 21/71
8/8 [==============================] - 4s 526ms/step - loss: 0.2710 - accuracy: 0.8992 - val_loss: 0.2314 - val_accuracy: 0.8992
Epoch 22/71
8/8 [==============================] - 6s 793ms/step - loss: 0.2411 - accuracy: 0.9160 - val_loss: 0.1046 - val_accuracy: 0.9748
Epoch 23/71
8/8 [==============================] - 4s 520ms/step - loss: 0.2236 - accuracy: 0.9244 - val_loss: 0.1046 - val_accuracy: 0.9748
Epoch 24/71
8/8 [==============================] - 6s 717ms/step - loss: 0.2658 - accuracy: 0.8739 - val_loss: 0.1743 - val_accuracy: 0.9412
Epoch 25/71
8/8 [==============================] - 4s 576ms/step - loss: 0.3596 - accuracy: 0.8655 - val_loss: 0.1137 - val_accuracy: 0.9580
Epoch 26/71
8/8 [==============================] - 5s 616ms/step - loss: 0.2370 - accuracy: 0.8908 - val_loss: 0.1398 - val_accuracy: 0.9412
Epoch 27/71
8/8 [==============================] - 4s 569ms/step - loss: 0.2839 - accuracy: 0.9244 - val_loss: 0.1127 - val_accuracy: 0.9664
Epoch 28/71
```

```
epochs_range = range(EPOCHS)
```

```
epochs_range
```

```
range(0, 71)
```

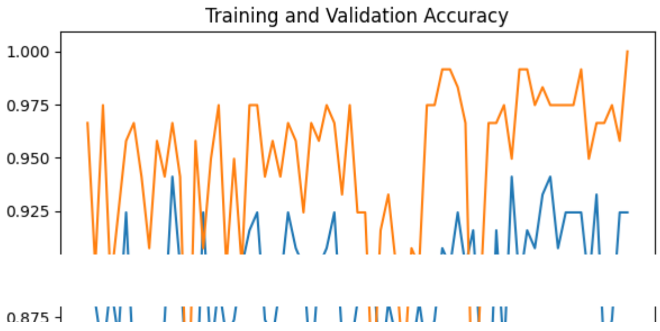
```
# plot epoch 71 accuracy ad loss :
```

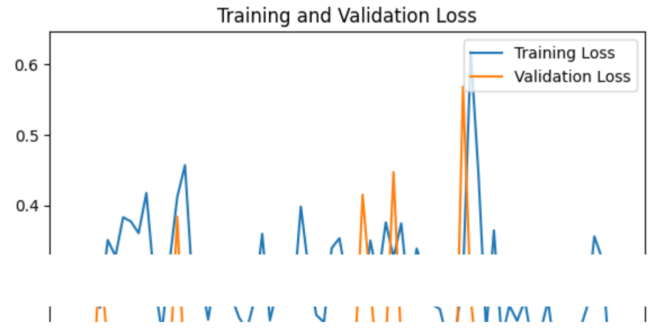
```
# Plot training and validation graphs
acc = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

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

```
acc
val_accuracy
```

```
[0.9663865566253662,
 0.8991596698760986,
 0.9747899174690247,
 0.8907563090324402,
 0.924369752407074,
 0.9579831957817078,
 0.9663865566253662,
 0.9411764740943909,
 0.9075630307197571,
 0.9579831957817078,
 0.9411764740943909,
 0.9663865566253662,
 0.9411764740943909,
 0.8151260614395142,
 0.9579831957817078,
 0.9075630307197571,
 0.9495798349380493,
 0.9747899174690247,
 0.8991596698760986,
 0.9495798349380493,
 0.8991596698760986,
 0.9747899174690247,
 0.9747899174690247,
 0.9411764740943909,
 0.9579831957817078,
 0.9411764740943909,
 0.9663865566253662,
 0.9579831957817078,
 0.924369752407074,
 0.9663865566253662,
 0.9579831957817078,
 0.9747899174690247,
 0.9663865566253662,
 0.9327731132507324,
 0.9747899174690247,
 0.924369752407074,
 0.924369752407074,
 0.8403361439704895,
 0.9159663915634155,
 0.9327731132507324,
 0.8991596698760986,
 0.848739504814148,
 0.9075630307197571,
 0.8991596698760986,
 0.9747899174690247,
 0.9747899174690247,
 0.9915966391563416,
 0.9915966391563416,
 0.9831932783126831,
 0.9663865566253662,
 0.8151260614395142,
 0.8907563090324402,
 0.9663865566253662,
 0.9663865566253662,
 0.9747899174690247,
 0.9495798349380493,
 0.9915966391563416,
 0.9915966391563416,
```

```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.plot(epochs_range,acc,label='Training Accuracy')
plt.plot(epochs_range,val_accuracy,label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1,2,2)
plt.plot(epochs_range,loss,label='Training Loss')
plt.plot(epochs_range,val_loss,label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```





5.SAVING DEEP LEARNING MODEL

```
model.save('final_model1.h5')
```

6.conclusions: *Three different classifications of rice leaf diseases were studied and compared using the convolutional neural network. Using more than 5000 images as training data we were able to identify what type of disease a leaf has with 80.2% accuracy. Diving deeper, we can accurately classify acauracy and loss.

For future studies, we recommend to improve the accuracy of prediction by exploring other network architecture or adding layers or nodes to the current model, scale the project to other diseases for the model to be more general, and, lastly, consult with farmers and consider their input into the pre-processing and deployment of the model.*

✓ 0s completed at 4:15 AM