Potato Disease Classification Using Neural Network

Every year, farmers suffer economic losses and crop waste owing to different illnesses in potato plants. We will utilise CNN to classify images and create a smartphone app that will allow a farmer to take a picture of a plant and determine whether or not it has a disease. This project will use the following technology stack:

Model Construction: CNN, TensorFlow, Data Augmentation, and TF Dataset.

1) The project includes data collection, model building using Convolutional Neural Network (CNN), ML ops using TF serving, a backend server using Fast API, deployment to Google Cloud, and a mobile app in React Native.

2) The main problem addressed is the economic losses faced by potato farmers due to early blight and late blight diseases.

3) The application will classify images of potato plants as healthy or having early blight or late blight.

4) The data collection process involves gathering images of healthy potato plant leaves and leaves with early blight or late blight.

5) Data cleaning and pre-processing will be done using tf dataset and data augmentation.

6) Model building will be carried out using CNN, and the trained model will be exported onto the disk.

IMPORTNG LIBRARIES

```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
import pandas as pd
import numpy
```

```
WARNING:tensorflow:From c:\Program Files\Python311\Lib\site-packages\
keras\src\losses.py:2976: The name
tf.losses.sparse_softmax_cross_entropy is deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

```
IMAGE_SIZE = 256
BATCH = 32
CHANNELS = 3
EPOCHS = 50
```

CONVERTING IMAGES TO TENSORS

```
''' For downloading dataset into tensorfolw dataset using keras '''
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    r"C:\Users\raman\Desktop\mini-projects\potato class\potato",
     shuffle = True,
     image_size = (IMAGE_SIZE,IMAGE_SIZE),
     batch_size = BATCH)
```

```
Found 2152 files belonging to 3 classes.
```

```
class_names = dataset.class_names
class_names    # for printing class names
```

```
['Potato___Early_blight', 'Potato___Late_blight', 'Potato___healthy']
```

```
len(dataset)  # 68 -> represent number of batches so 68*32 will give
total length
```

```
68
```

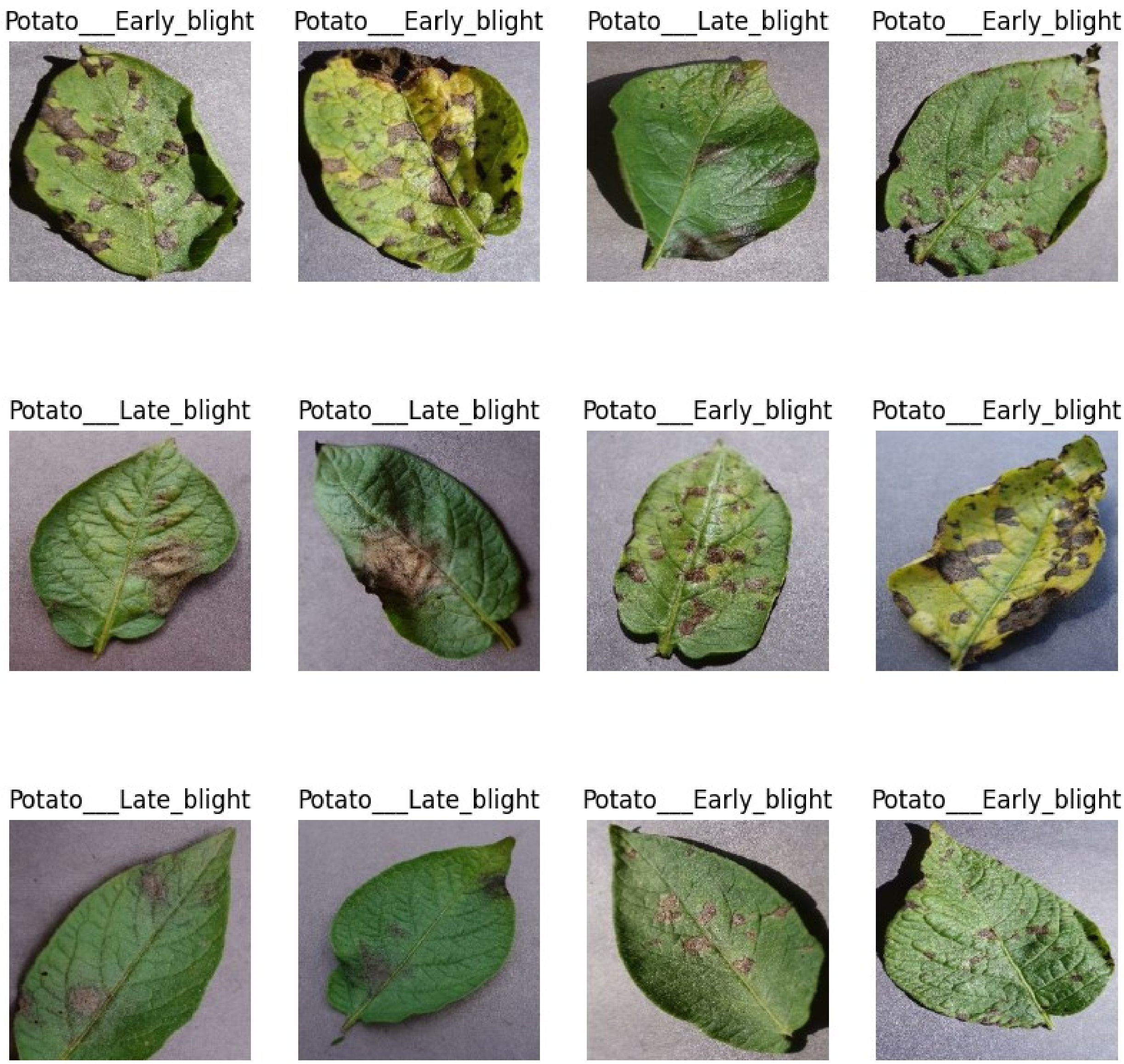
```
for image_batch,label_batch in dataset.take(1):
    print(image_batch.shape)
    print(label_batch.numpy())

'''
32 -> batches
256,256 -> dimensions
3 -> RGB
'''
```

```
(32, 256, 256, 3)
[1 0 2 0 1 0 1 0 1 1 0 1 1 1 1 1 1 1 0 0 2 1 0 0 0 0 0 0 1 1 1 1]
```

```
'\n32 -> batches\n256,256 -> dimensions\n3 -> RGB\n'
```

```
plt.figure(figsize=(10,10))
for image_batch,label_batch in dataset.take(1):
    for i in range(12):
        plt.subplot(3,4,i+1)
        plt.imshow(image_batch[i].numpy().astype('uint8'))
        plt.title(class_names[label_batch[i]])
        plt.axis('off')

'''
plt.imshow()-> to represent matrix into picture
'''
```

```
'\nplt.imshow()-> to represent matrix into picture\n'
```



DATA SPLITTING

```
'''
training = 80% data
validation = 10% data
test = 10% data
'''

'\ntraining = 80% data\nvalidation = 10% data\ntest = 10% data\n'

def
get_dataset_partition_tf(ds,train_split=0.8,val_split=0.1,test_split=0
.1,shuffle=True,shuffle_size=10000):
```

```
    '''
    For Splitting data into training, validation and testing data
    '''
    ds_size = len(ds)
    if shuffle:
        ds = ds.shuffle(shuffle_size,seed=12)
    train_size = int(train_split*ds_size)
    val_size = int(val_split*ds_size)

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)
    return train_ds , val_ds , test_ds

train_ds , val_ds , test_ds = get_dataset_partition_tf(dataset)

len(train_ds) , len(val_ds) , len(test_ds)

(54, 6, 8)

##

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size =
tf.data.AUTOTUNE)
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size =
tf.data.AUTOTUNE)
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size =
tf.data.AUTOTUNE)
```

PREPROCESSING

```
resize_and_rescale = tf.keras.Sequential([
    layers.experimental.preprocessing.Resizing(IMAGE_SIZE,IMAGE_SIZE),
    layers.experimental.preprocessing.Rescaling(1.0/255)
]) ### Adding Layer for rescaling and resizing the data

WARNING:tensorflow:From c:\Program Files\Python311\Lib\site-packages\
keras\src\backend.py:873: The name tf.get_default_graph is deprecated.
Please use tf.compat.v1.get_default_graph instead.
```

DATA AUGMENTATION

```
data_augmentation = tf.keras.Sequential([

layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"
),
    layers.experimental.preprocessing.RandomRotation(0.2)
])

## Adding 2 Data augmentation filters RandomFlip and RandomRotation
```

MODELLING USING CONVOLUTIONAL NEURAL NETWORK

```
'''
(convolutional+ReLu --> Pooling)n --> Dense Layer
Pooling :-
        - MAX Pooling
'''
```

```
'\n(convolutional+ReLu --> Pooling)n --> Dense Layer\nPooling :-\n
- MAX Pooling\n'
```

Layers. Conv2D(Filter , kernel , activation , input_shape) -> first layer

```
    Polling2D(kernel)

    Conv2D(filter, kernel, activation)
```

```
input_shape = (BATCH,IMAGE_SIZE,IMAGE_SIZE,CHANNELS)
n_classes = 3
model = models.Sequential([
    resize_and_rescale,
    data_augmentation,
    layers.Conv2D(32,(3,3),activation="relu",input_shape =
input_shape),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64,kernel_size = (3,3),activation="relu"),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64,kernel_size = (3,3),activation = "relu"),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64,kernel_size = (3,3),activation = "relu"),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64,kernel_size = (3,3),activation = "relu"),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64,kernel_size = (3,3),activation = "relu"),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(64,activation = "relu"),
    layers.Dense(n_classes,activation = "softmax")
])

model.build(input_shape = input_shape)
```

```
WARNING:tensorflow:From c:\Program Files\Python311\Lib\site-packages\
keras\src\layers\pooling\max_pooling2d.py:161: The name tf.nn.max_pool
is deprecated. Please use tf.nn.max_pool2d instead.
```

```
model.summary()  # summary of neural network architecture
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 sequential (Sequential)     (32, 256, 256, 3)         0         
                                                                 
 sequential_1 (Sequential)   (32, 256, 256, 3)         0         
                                                                 
 conv2d (Conv2D)             (32, 254, 254, 32)        896       
                                                                 
 max_pooling2d (MaxPooling2  (32, 127, 127, 32)        0         
 D)                                                              
                                                                 
 conv2d_1 (Conv2D)           (32, 125, 125, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPoolin  (32, 62, 62, 64)          0         
 g2D)                                                            
                                                                 
 conv2d_2 (Conv2D)           (32, 60, 60, 64)          36928     
                                                                 
 max_pooling2d_2 (MaxPoolin  (32, 30, 30, 64)          0         
 g2D)                                                            
                                                                 
 conv2d_3 (Conv2D)           (32, 28, 28, 64)          36928     
                                                                 
 max_pooling2d_3 (MaxPoolin  (32, 14, 14, 64)          0         
 g2D)                                                            
                                                                 
 conv2d_4 (Conv2D)           (32, 12, 12, 64)          36928     
                                                                 
 max_pooling2d_4 (MaxPoolin  (32, 6, 6, 64)            0         
 g2D)                                                            
                                                                 
 conv2d_5 (Conv2D)           (32, 4, 4, 64)            36928     
                                                                 
 max_pooling2d_5 (MaxPoolin  (32, 2, 2, 64)            0         
 g2D)                                                            
                                                                 
 flatten (Flatten)           (32, 256)                 0         
                                                                 
 dense (Dense)               (32, 64)                  16448     
                                                                 
 dense_1 (Dense)             (32, 3)                   195       
                                                                 
=================================================================
Total params: 183747 (717.76 KB)
Trainable params: 183747 (717.76 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.compile(
    optimizer = 'adam',
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits =
False),
    metrics = ['accuracy']
)
## keras.losses.SparseCategoricalCrossentropy
```

```
WARNING:tensorflow:From c:\Program Files\Python311\Lib\site-packages\
keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is
deprecated. Please use tf.compat.v1.train.Optimizer instead.
```

```
history = model.fit(
    train_ds,
    epochs = EPOCHS,
    batch_size = BATCH,
    verbose = 1,
    validation_data = val_ds
)
```

```
Epoch 1/50
```

```
WARNING:tensorflow:From c:\Program Files\Python311\Lib\site-packages\
keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue
is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue
instead.
```

```
WARNING:tensorflow:From c:\Program Files\Python311\Lib\site-packages\
keras\src\engine\base_layer_utils.py:384: The name
tf.executing_eagerly_outside_functions is deprecated. Please use
tf.compat.v1.executing_eagerly_outside_functions instead.
```

```
54/54 [==============================] - 64s 1s/step - loss: 0.8222 -
accuracy: 0.5885 - val_loss: 0.5505 - val_accuracy: 0.7292
Epoch 2/50
54/54 [==============================] - 50s 931ms/step - loss: 0.4443
- accuracy: 0.8137 - val_loss: 0.6161 - val_accuracy: 0.7188
Epoch 3/50
54/54 [==============================] - 50s 933ms/step - loss: 0.3319
- accuracy: 0.8571 - val_loss: 0.3821 - val_accuracy: 0.8802
Epoch 4/50
54/54 [==============================] - 49s 918ms/step - loss: 0.2503
- accuracy: 0.9028 - val_loss: 0.3420 - val_accuracy: 0.8646
Epoch 5/50
54/54 [==============================] - 50s 926ms/step - loss: 0.1589
- accuracy: 0.9369 - val_loss: 0.6096 - val_accuracy: 0.8073
Epoch 6/50
54/54 [==============================] - 47s 871ms/step - loss: 0.6019
- accuracy: 0.7315 - val_loss: 0.3622 - val_accuracy: 0.8438
```

```
Epoch 7/50
54/54 [==============================] - 42s 788ms/step - loss: 0.2667
- accuracy: 0.8918 - val_loss: 0.5683 - val_accuracy: 0.7604
Epoch 8/50
54/54 [==============================] - 43s 790ms/step - loss: 0.1655
- accuracy: 0.9346 - val_loss: 0.1595 - val_accuracy: 0.9375
Epoch 9/50
54/54 [==============================] - 42s 775ms/step - loss: 0.1444
- accuracy: 0.9427 - val_loss: 0.1470 - val_accuracy: 0.9427
Epoch 10/50
54/54 [==============================] - 39s 718ms/step - loss: 0.1324
- accuracy: 0.9410 - val_loss: 0.1416 - val_accuracy: 0.9427
Epoch 11/50
54/54 [==============================] - 38s 701ms/step - loss: 0.1879
- accuracy: 0.9207 - val_loss: 0.2019 - val_accuracy: 0.9271
Epoch 12/50
54/54 [==============================] - 38s 698ms/step - loss: 0.0972
- accuracy: 0.9653 - val_loss: 0.2838 - val_accuracy: 0.9115
Epoch 13/50
54/54 [==============================] - 36s 669ms/step - loss: 0.0743
- accuracy: 0.9745 - val_loss: 0.4919 - val_accuracy: 0.8385
Epoch 14/50
54/54 [==============================] - 36s 671ms/step - loss: 0.1457
- accuracy: 0.9444 - val_loss: 0.4919 - val_accuracy: 0.7865
Epoch 15/50
54/54 [==============================] - 37s 676ms/step - loss: 0.1305
- accuracy: 0.9508 - val_loss: 0.1223 - val_accuracy: 0.9531
Epoch 16/50
54/54 [==============================] - 36s 670ms/step - loss: 0.0603
- accuracy: 0.9797 - val_loss: 0.1427 - val_accuracy: 0.9583
Epoch 17/50
54/54 [==============================] - 36s 670ms/step - loss: 0.0517
- accuracy: 0.9803 - val_loss: 0.7570 - val_accuracy: 0.8021
Epoch 18/50
54/54 [==============================] - 36s 675ms/step - loss: 0.0610
- accuracy: 0.9769 - val_loss: 0.1635 - val_accuracy: 0.9479
Epoch 19/50
54/54 [==============================] - 39s 724ms/step - loss: 0.0425
- accuracy: 0.9832 - val_loss: 0.1561 - val_accuracy: 0.9427
Epoch 20/50
54/54 [==============================] - 37s 677ms/step - loss: 0.0527
- accuracy: 0.9844 - val_loss: 0.1201 - val_accuracy: 0.9688
Epoch 21/50
54/54 [==============================] - 37s 692ms/step - loss: 0.0491
- accuracy: 0.9838 - val_loss: 0.0357 - val_accuracy: 0.9896
Epoch 22/50
54/54 [==============================] - 37s 682ms/step - loss: 0.0569
- accuracy: 0.9792 - val_loss: 0.0697 - val_accuracy: 0.9688
Epoch 23/50
```

```
54/54 [==============================] - 37s 679ms/step - loss: 0.0495
- accuracy: 0.9815 - val_loss: 0.2513 - val_accuracy: 0.9375
Epoch 24/50
54/54 [==============================] - 37s 691ms/step - loss: 0.0692
- accuracy: 0.9745 - val_loss: 0.0565 - val_accuracy: 0.9792
Epoch 25/50
54/54 [==============================] - 38s 694ms/step - loss: 0.0506
- accuracy: 0.9838 - val_loss: 0.1417 - val_accuracy: 0.9635
Epoch 26/50
54/54 [==============================] - 37s 680ms/step - loss: 0.0551
- accuracy: 0.9780 - val_loss: 0.0705 - val_accuracy: 0.9740
Epoch 27/50
54/54 [==============================] - 38s 703ms/step - loss: 0.0364
- accuracy: 0.9884 - val_loss: 0.3109 - val_accuracy: 0.9323
Epoch 28/50
54/54 [==============================] - 43s 801ms/step - loss: 0.0350
- accuracy: 0.9884 - val_loss: 0.1509 - val_accuracy: 0.9688
Epoch 29/50
54/54 [==============================] - 46s 855ms/step - loss: 0.0767
- accuracy: 0.9745 - val_loss: 0.0935 - val_accuracy: 0.9792
Epoch 30/50
54/54 [==============================] - 45s 827ms/step - loss: 0.0599
- accuracy: 0.9774 - val_loss: 0.2052 - val_accuracy: 0.9427
Epoch 31/50
54/54 [==============================] - 44s 810ms/step - loss: 0.0490
- accuracy: 0.9803 - val_loss: 0.1339 - val_accuracy: 0.9688
Epoch 32/50
54/54 [==============================] - 51s 941ms/step - loss: 0.0413
- accuracy: 0.9850 - val_loss: 0.2492 - val_accuracy: 0.9479
Epoch 33/50
54/54 [==============================] - 46s 858ms/step - loss: 0.0531
- accuracy: 0.9844 - val_loss: 0.0718 - val_accuracy: 0.9792
Epoch 34/50
54/54 [==============================] - 43s 792ms/step - loss: 0.0375
- accuracy: 0.9884 - val_loss: 0.1556 - val_accuracy: 0.9583
Epoch 35/50
54/54 [==============================] - 44s 807ms/step - loss: 0.0547
- accuracy: 0.9826 - val_loss: 0.0741 - val_accuracy: 0.9688
Epoch 36/50
54/54 [==============================] - 53s 990ms/step - loss: 0.0731
- accuracy: 0.9740 - val_loss: 0.2379 - val_accuracy: 0.9375
Epoch 37/50
54/54 [==============================] - 45s 833ms/step - loss: 0.0580
- accuracy: 0.9809 - val_loss: 0.0923 - val_accuracy: 0.9688
Epoch 38/50
54/54 [==============================] - 49s 910ms/step - loss: 0.0223
- accuracy: 0.9931 - val_loss: 0.0747 - val_accuracy: 0.9792
Epoch 39/50
54/54 [==============================] - 52s 959ms/step - loss: 0.0537
```

```
- accuracy: 0.9838 - val_loss: 0.1442 - val_accuracy: 0.9479
Epoch 40/50
54/54 [==============================] - 53s 979ms/step - loss: 0.0471
- accuracy: 0.9838 - val_loss: 0.0696 - val_accuracy: 0.9792
Epoch 41/50
54/54 [==============================] - 56s 1s/step - loss: 0.0298 -
accuracy: 0.9919 - val_loss: 0.0947 - val_accuracy: 0.9688
Epoch 42/50
54/54 [==============================] - 51s 956ms/step - loss: 0.0344
- accuracy: 0.9873 - val_loss: 0.2150 - val_accuracy: 0.9427
Epoch 43/50
54/54 [==============================] - 42s 782ms/step - loss: 0.0820
- accuracy: 0.9716 - val_loss: 0.0477 - val_accuracy: 0.9844
Epoch 44/50
54/54 [==============================] - 44s 809ms/step - loss: 0.0455
- accuracy: 0.9838 - val_loss: 0.0737 - val_accuracy: 0.9792
Epoch 45/50
54/54 [==============================] - 41s 766ms/step - loss: 0.0225
- accuracy: 0.9942 - val_loss: 0.1055 - val_accuracy: 0.9688
Epoch 46/50
54/54 [==============================] - 37s 674ms/step - loss: 0.0351
- accuracy: 0.9884 - val_loss: 0.2235 - val_accuracy: 0.9427
Epoch 47/50
54/54 [==============================] - 40s 745ms/step - loss: 0.0375
- accuracy: 0.9873 - val_loss: 0.5801 - val_accuracy: 0.8438
Epoch 48/50
54/54 [==============================] - 38s 696ms/step - loss: 0.0265
- accuracy: 0.9890 - val_loss: 0.0977 - val_accuracy: 0.9688
Epoch 49/50
54/54 [==============================] - 36s 667ms/step - loss: 0.0338
- accuracy: 0.9878 - val_loss: 0.2137 - val_accuracy: 0.9583
Epoch 50/50
54/54 [==============================] - 37s 689ms/step - loss: 0.0304
- accuracy: 0.9867 - val_loss: 0.0861 - val_accuracy: 0.9740
```

```
scores = model.evaluate(test_ds)
```

```
8/8 [==============================] - 7s 255ms/step - loss: 0.0516 -
accuracy: 0.9766
```

```
scores
```

```
[0.05161502584815025, 0.9765625]
```

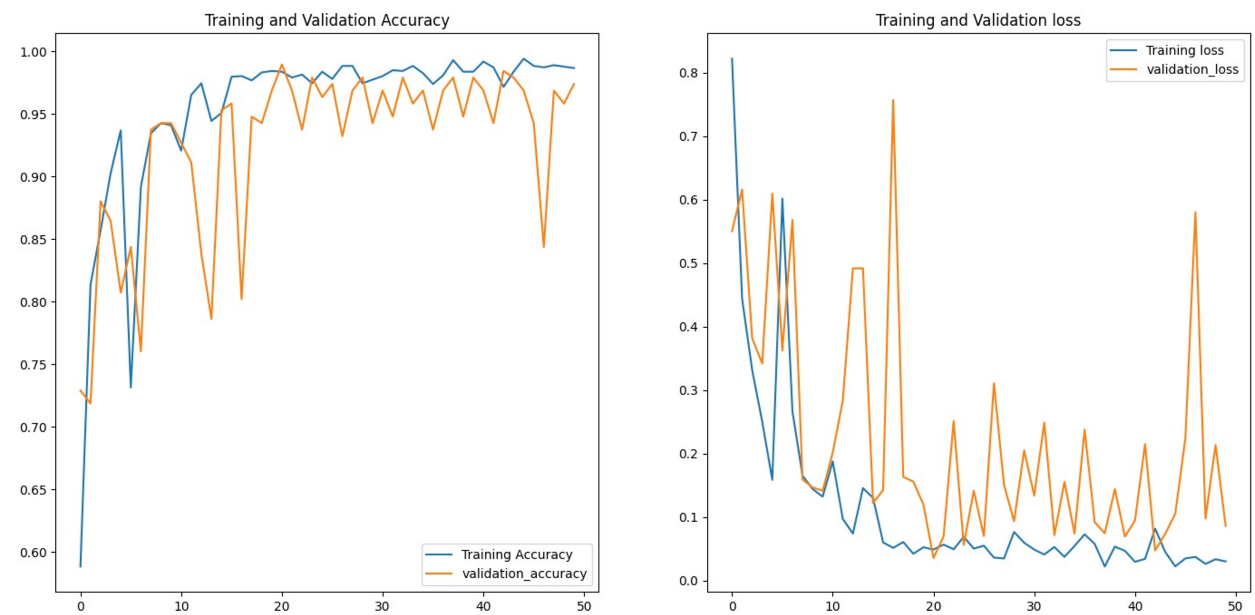
```
history
```

```
<keras.src.callbacks.History at 0x24bfcc52b50>
```

```
history.params
```

```
{'verbose': 1, 'epochs': 50, 'steps': 54}
```

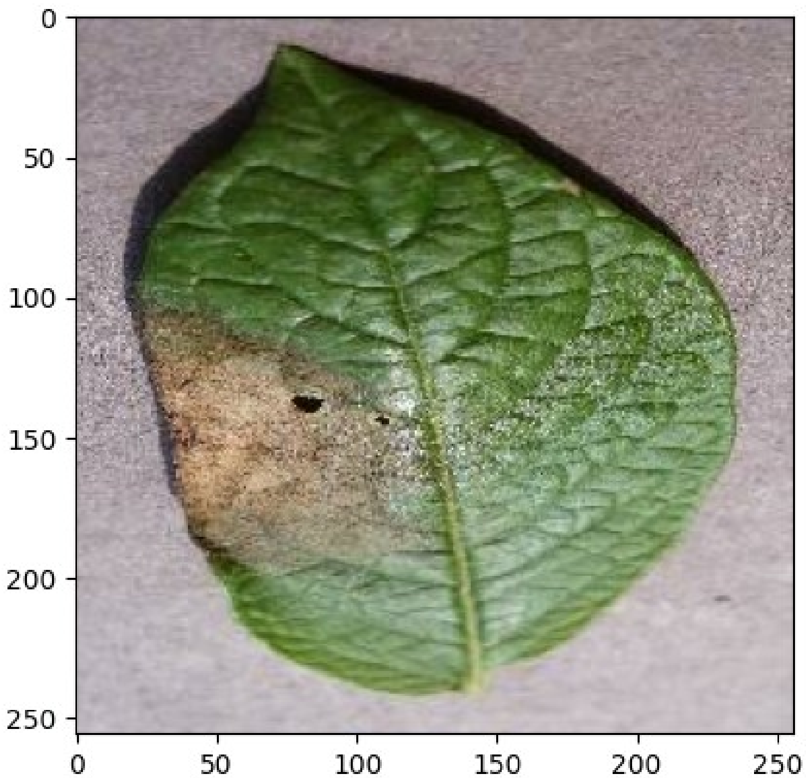
```
history.history.keys()

dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])

acc = history.history["accuracy"]
val_acc= history.history["val_accuracy"]

loss = history.history["loss"]
val_loss = history.history["val_loss"]

plt.figure(figsize=(17,8))
plt.subplot(1,2,1)
plt.plot(range(EPOCHS),acc,label="Training Accuracy")
plt.plot(range(EPOCHS),val_acc,label = "validation_accuracy")
plt.legend(loc="lower right")
plt.title("Training and Validation Accuracy")

plt.subplot(1,2,2)
plt.plot(range(EPOCHS),loss,label="Training loss")
plt.plot(range(EPOCHS),val_loss,label = "validation_loss")
plt.legend(loc="upper right")
plt.title("Training and Validation loss")

Text(0.5, 1.0, 'Training and Validation loss')
```




```
for image_batch,label_batch in test_ds.take(1):
    first_image = image_batch[0].numpy().astype("uint8")
    first_label = label_batch[0].numpy()

    print('first image to predict')
    plt.imshow(first_image)
```

```
    print('actual label : ',class_names[first_label])

    batch_prediction = model.predict(image_batch)
    print(class_names[numpy.argmax(batch_prediction[0])])
```

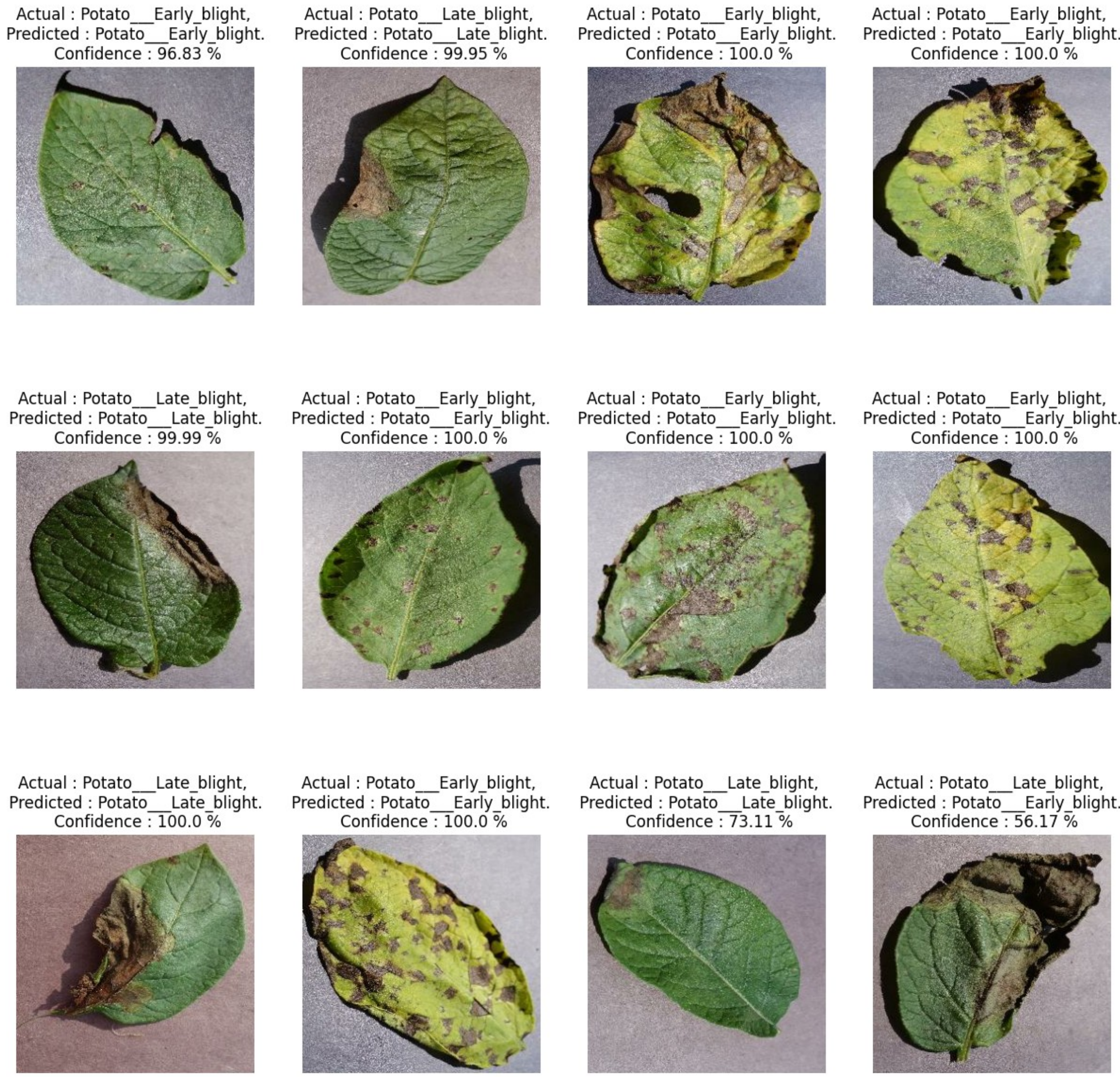
```
first image to predict
actual label :  Potato___Late_blight
1/1 [==============================] - 1s 817ms/step
Potato___Late_blight
```




```
## Defining functions for predicting test dataset results

def predict(model,img):
    img_arry = tf.keras.preprocessing.image.img_to_array((img))
    img_array = tf.expand_dims(img_arry,0)  ## creating a batch
    # prediction
    prediction = model.predict(img_array)

    pred_class = class_names[numpy.argmax(prediction[0])]
    confidence = round(100*(numpy.max(prediction[0])),2)
    return pred_class , confidence
```

```
plt.figure(figsize = (15,15))
for images,label in test_ds.take(1):
    for i in range(12):
        ax = plt.subplot(3,4,i+1)
        plt.imshow(images[i].numpy().astype('uint8'))

        pred_class , confidence = predict(model,images[i])
        actual_class = class_names[label[i]]

        plt.title(f"Actual : {actual_class}, \n Predicted :
{pred_class}. \n Confidence : {confidence} %")
        plt.axis('off')
```

```
1/1 [==============================] - 0s 348ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 54ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 53ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 49ms/step
```




```
import os
model_version = max([int(i) for i in os.listdir(r"C:\Users\raman\
Desktop\mini-projects\potato class\models") + [0]])+1
model.save(f'Potato Disease {model_version}')
```

```
INFO:tensorflow:Assets written to: Potato Disease 1\assets
```



```
INFO:tensorflow:Assets written to: Potato Disease 1\assets
```