Potato Leaf Dataset Pakistan

Transfer Learning models (VGG19, GoogLeNet, ResNet50)

Inspiration taken from: https://www.kaggle.com/code/sakthimurugavel/potato-leaf-disease-classifier-review-ii-final/notebook

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

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

Import all the required dependencies

```
import os
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf

from tensorflow.keras import models, layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import load_model, Model
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.resnet_v2 import ResNet50V2
from tensorflow.keras.utils import plot_model
```

Define the constants

```
BATCH_SIZE = 32
IMAGE_SIZE = 224
CHANNELS = 3
EPOCHS = 50
```

```
path = "/content/drive/MyDrive/Colab Notebooks/Dissertation/Code/PLD_larger_dataset"
```

```
dataset = tf.keras.preprocessing.image_dataset_from_directory(path,
                                                              shuffle = True,
                                                              image_size = (IMAGE_SIZE, IMAGE_SIZE),
                                                              batch_size = BATCH_SIZE
)
```

```
Found 4072 files belonging to 3 classes.
```

```
class_names = dataset.class_names
class_names # 0 = early blight, 1 = late blight, 2 = healthy
```

```
['Potato_Early_blight', 'Potato_Late_blight', 'Potato_healthy']
```

```
def DatasetSize(path):
    number_of_images = {} #tuple
    for folder in os.listdir(path):
        number_of_images[folder] = len(os.listdir(os.path.join(path, folder)))
    return number_of_images

data_set = DatasetSize(path)
print(data_set)
```

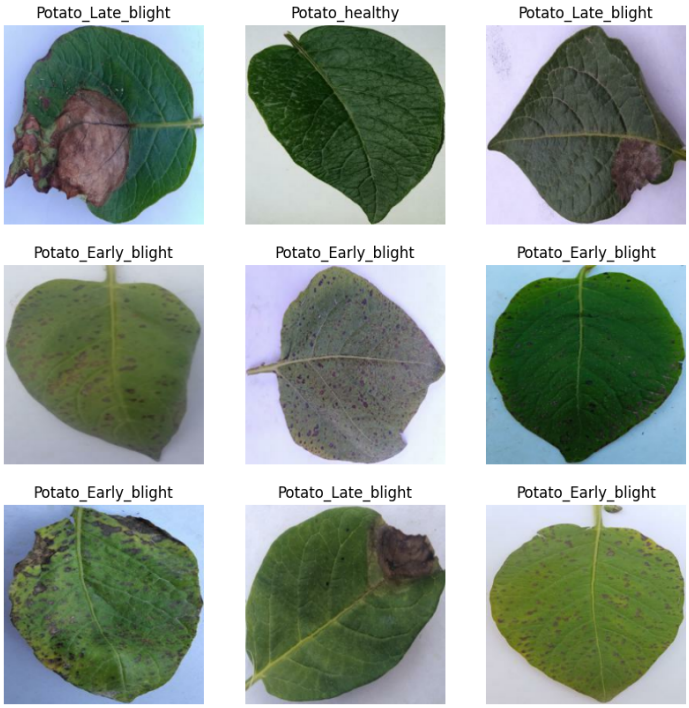
```
{'Potato_Early_blight': 1628, 'Potato_healthy': 1020, 'Potato_Late_blight': 1424}
```

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

```
(32, 224, 224, 3)
[1 2 2 0 0 0 1 1 1 1 2 0 1 2 2 0 2 0 1 0 2 2 0 1 2 0 2 2 0 2 1 1]
```

Visualise the images

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



Split the dataset

```
# Use split folders
#splitfolders --ratio .8 .1 .1 --
```

```
!pip install split-folders
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting split-folders
  Downloading split_folders-0.5.1-py3-none-any.whl (8.4 kB)
Installing collected packages: split-folders
Successfully installed split-folders-0.5.1
```

```
import splitfolders
splitfolders.ratio(path, output="/content/drive/MyDrive/Colab Notebooks/Dissertation/Code/TransferLearning/outputlarge", seed=1337, ratio
```

```
Copying files: 4072 files [26:25,  2.57 files/s]
```

Data Preprocessing and Augmentation

```
train_datagen = ImageDataGenerator(
        rescale = 1./255,
        rotation_range = 10,
        horizontal_flip = True
)

train_generator = train_datagen.flow_from_directory("/content/drive/MyDrive/Colab Notebooks/Dissertation/Code/TransferLearning/outputlarg
                                                    target_size = (IMAGE_SIZE, IMAGE_SIZE),
                                                    batch_size = 32,
                                                    class_mode = "sparse")
```

```
Found 3257 images belonging to 3 classes.
```

```
train_generator.class_indices
```

```
{'Potato_Early_blight': 0, 'Potato_Late_blight': 1, 'Potato_healthy': 2}
```

```
class_names = list(train_generator.class_indices.keys())
class_names
```

```
['Potato_Early_blight', 'Potato_Late_blight', 'Potato_healthy']
```

```
# Validation

validation_datagen = ImageDataGenerator(
        rescale = 1./255,
        rotation_range = 10,
        horizontal_flip = True
)

validation_generator = validation_datagen.flow_from_directory("/content/drive/MyDrive/Colab Notebooks/Dissertation/Code/TransferLearning/
                                                    target_size = (IMAGE_SIZE, IMAGE_SIZE),
                                                    batch_size = 32,
                                                    class_mode = "sparse")
```

```
Found 406 images belonging to 3 classes.
```

```
# Test

test_datagen = ImageDataGenerator(
        rescale = 1./255,
        rotation_range = 10,
        horizontal_flip = True
)

test_generator = test_datagen.flow_from_directory("/content/drive/MyDrive/Colab Notebooks/Dissertation/Code/TransferLearning/outputlarge/
                                                    target_size = (IMAGE_SIZE, IMAGE_SIZE),
                                                    batch_size = 32,
                                                    class_mode = "sparse")
```

```
Found 409 images belonging to 3 classes.
```

VGG19

```
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 3

def create_model():
    model = models.Sequential()
    model.add(VGG19(include_top = False, weights = 'imagenet', input_shape = (IMAGE_SIZE, IMAGE_SIZE, CHANNELS)))
    model.add(layers.GlobalAveragePooling2D())
    model.add(layers.Flatten())
    model.add(layers.Dense(256, activation = 'relu'))
    model.add(layers.Dropout(0.3))
    model.add(layers.Dense(n_classes, activation='softmax'))

    return model


vgg19_with_aug_model = create_model()
vgg19_with_aug_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_n
80134624/80134624 [==============================] - 3s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
 vgg19 (Functional)          (None, 7, 7, 512)         20024384

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 256)               131328

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 3)                 771

=================================================================
Total params: 20,156,483
Trainable params: 20,156,483
Non-trainable params: 0
_________________________________________________________________
```

Compiling the model and determining the accuracy

```
vgg19_with_aug_model.compile(
                           optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
                           loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
                           metrics=['accuracy'])


vgg19_with_aug_history = vgg19_with_aug_model.fit(
    train_generator,
    steps_per_epoch=54,
    batch_size=32,
    validation_data=validation_generator,
    validation_steps=7,
    verbose=1,
    epochs=EPOCHS,
)
```

```
Epoch 1/50
54/54 [==============================] - 50s 557ms/step - loss: 1.1047 - accuracy: 0.3696 - val_loss: 1.0822 - val_accuracy: 0.4
Epoch 2/50
54/54 [==============================] - 26s 488ms/step - loss: 1.0753 - accuracy: 0.3951 - val_loss: 1.0187 - val_accuracy: 0.4
Epoch 3/50
54/54 [==============================] - 26s 478ms/step - loss: 0.9861 - accuracy: 0.4753 - val_loss: 0.8455 - val_accuracy: 0.6
Epoch 4/50
54/54 [==============================] - 28s 522ms/step - loss: 0.7941 - accuracy: 0.6153 - val_loss: 0.6704 - val_accuracy: 0.7
Epoch 5/50
54/54 [==============================] - 27s 502ms/step - loss: 0.6613 - accuracy: 0.7130 - val_loss: 0.5951 - val_accuracy: 0.7
Epoch 6/50
54/54 [==============================] - 26s 476ms/step - loss: 0.5416 - accuracy: 0.7780 - val_loss: 0.3680 - val_accuracy: 0.8
Epoch 7/50
54/54 [==============================] - 26s 475ms/step - loss: 0.3831 - accuracy: 0.8565 - val_loss: 0.4552 - val_accuracy: 0.8
Epoch 8/50
54/54 [==============================] - 26s 485ms/step - loss: 0.2982 - accuracy: 0.9057 - val_loss: 0.3982 - val_accuracy: 0.8
Epoch 9/50
54/54 [==============================] - 26s 477ms/step - loss: 0.2692 - accuracy: 0.9010 - val_loss: 0.1518 - val_accuracy: 0.9
Epoch 10/50
54/54 [==============================] - 26s 482ms/step - loss: 0.1539 - accuracy: 0.9512 - val_loss: 0.2525 - val_accuracy: 0.9
Epoch 11/50
54/54 [==============================] - 26s 488ms/step - loss: 0.1196 - accuracy: 0.9634 - val_loss: 0.1014 - val_accuracy: 0.9
Epoch 12/50
54/54 [==============================] - 26s 482ms/step - loss: 0.1285 - accuracy: 0.9669 - val_loss: 0.0656 - val_accuracy: 0.9
Epoch 13/50
54/54 [==============================] - 26s 486ms/step - loss: 0.0806 - accuracy: 0.9709 - val_loss: 0.1211 - val_accuracy: 0.9
Epoch 14/50
54/54 [==============================] - 26s 482ms/step - loss: 0.0588 - accuracy: 0.9838 - val_loss: 0.0856 - val_accuracy: 0.9
Epoch 15/50
54/54 [==============================] - 26s 481ms/step - loss: 0.0783 - accuracy: 0.9780 - val_loss: 0.0951 - val_accuracy: 0.9
Epoch 16/50
54/54 [==============================] - 26s 478ms/step - loss: 0.0857 - accuracy: 0.9698 - val_loss: 0.1825 - val_accuracy: 0.9
Epoch 17/50
54/54 [==============================] - 26s 482ms/step - loss: 0.0580 - accuracy: 0.9826 - val_loss: 0.0948 - val_accuracy: 0.9
Epoch 18/50
54/54 [==============================] - 26s 484ms/step - loss: 0.0498 - accuracy: 0.9838 - val_loss: 0.1105 - val_accuracy: 0.9
Epoch 19/50
54/54 [==============================] - 27s 501ms/step - loss: 0.0983 - accuracy: 0.9705 - val_loss: 0.0533 - val_accuracy: 0.9
Epoch 20/50
54/54 [==============================] - 27s 497ms/step - loss: 0.0622 - accuracy: 0.9843 - val_loss: 0.0843 - val_accuracy: 0.9
Epoch 21/50
54/54 [==============================] - 26s 482ms/step - loss: 0.0698 - accuracy: 0.9779 - val_loss: 0.0379 - val_accuracy: 0.9
Epoch 22/50
54/54 [==============================] - 26s 480ms/step - loss: 0.0571 - accuracy: 0.9820 - val_loss: 0.0486 - val_accuracy: 0.9
Epoch 23/50
54/54 [==============================] - 26s 480ms/step - loss: 0.0507 - accuracy: 0.9884 - val_loss: 0.0813 - val_accuracy: 0.9
Epoch 24/50
```

```
54/54 [==============================] - 26s 482ms/step - loss: 0.0138 - accuracy: 0.9977 - val_loss: 0.0235 - val_accuracy: 0.99
Epoch 25/50
54/54 [==============================] - 26s 483ms/step - loss: 0.0628 - accuracy: 0.9831 - val_loss: 0.1178 - val_accuracy: 0.9
Epoch 26/50
54/54 [==============================] - 26s 487ms/step - loss: 0.1010 - accuracy: 0.9716 - val_loss: 0.0878 - val_accuracy: 0.9
Epoch 27/50
54/54 [==============================] - 26s 480ms/step - loss: 0.0338 - accuracy: 0.9919 - val_loss: 0.0971 - val_accuracy: 0.9
Epoch 28/50
54/54 [==============================] - 26s 489ms/step - loss: 0.0406 - accuracy: 0.9878 - val_loss: 0.0639 - val_accuracy: 0.9
Epoch 29/50
```

Producing the test accuracy and test loss

```
vgg19_with_aug_scores = vgg19_with_aug_model.evaluate(test_generator)
vgg19_with_aug_test_acc = vgg19_with_aug_scores[1]*100
vgg19_with_aug_test_loss = vgg19_with_aug_scores[0]
print("Test Loss: ", vgg19_with_aug_test_loss)
print("Test Accuracy: ", vgg19_with_aug_test_acc)
```

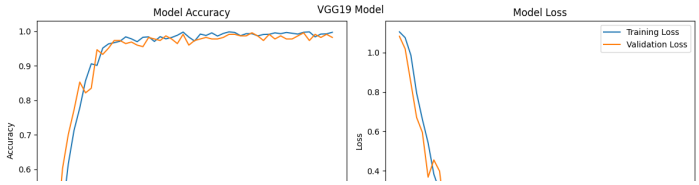
```
13/13 [==============================] - 5s 418ms/step - loss: 0.0078 - accuracy: 0.9976
Test Loss:  0.0077713532373309135
Test Accuracy:  99.75550174713135
```

VGG19 plot for accuracy vs loss

```
vgg19_with_aug_history
```

```
<keras.callbacks.History at 0x7fe330416c80>
```

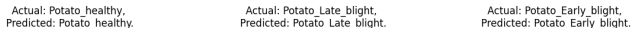
```
vgg19_with_aug_acc = vgg19_with_aug_history.history['accuracy']
vgg19_with_aug_val_acc = vgg19_with_aug_history.history['val_accuracy']

vgg19_with_aug_loss = vgg19_with_aug_history.history['loss']
vgg19_with_aug_val_loss = vgg19_with_aug_history.history['val_loss']

n = range(1, EPOCHS+1)


plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(n, vgg19_with_aug_acc, label='Training Accuracy')
plt.plot(n, vgg19_with_aug_val_acc, label='Validation Accuracy')
plt.legend(loc='best')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')

plt.subplot(1,2,2)
plt.plot(n, vgg19_with_aug_loss, label='Training Loss')
plt.plot(n, vgg19_with_aug_val_loss, label='Validation Loss')
plt.legend(loc='best')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Model Loss')
plt.tight_layout()
plt.suptitle('VGG19 Model')
plt.savefig("plot_vgg19.png")
plt.show()
```



Prediction of images using VGG19

```
def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i])
    img_array = tf.expand_dims(img_array, 0)

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence


plt.figure(figsize=(15, 15))
for images, labels in test_generator:
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i])

        predicted_class, confidence = predict(vgg19_with_aug_model, images[i])
        actual_class = class_names[int(labels[i])]

        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n Confidence: {confidence}%")

        plt.axis("off")
    break

plt.savefig("VGG19_prediction images")
plt.show()
```

```
1/1 [==============================] - 0s 463ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
```



GoogLeNet (InceptionV3)

```
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 3

def create_model():
    model = models.Sequential()
    model.add(InceptionV3(include_top = False, weights = 'imagenet', input_shape = (IMAGE_SIZE, IMAGE_SIZE, CHANNELS)))
    model.add(layers.GlobalAveragePooling2D())
    model.add(layers.Flatten())
    model.add(layers.Dense(256, activation = 'relu'))
    model.add(layers.Dropout(0.3))
    model.add(layers.Dense(n_classes, activation='softmax'))

    return model
```

```
inceptionv3_with_aug_model = create_model()
inceptionv3_with_aug_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_orderin
87910968/87910968 [==============================] - 3s 0us/step
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inception_v3 (Functional)   (None, 5, 5, 2048)        21802784

 global_average_pooling2d_1   (None, 2048)            0
 (GlobalAveragePooling2D)

 flatten_1 (Flatten)         (None, 2048)              0

 dense_2 (Dense)             (None, 256)               524544

 dropout_1 (Dropout)         (None, 256)               0

 dense_3 (Dense)             (None, 3)                 771

=================================================================
Total params: 22,328,099
Trainable params: 22,293,667
Non-trainable params: 34,432
_________________________________________________________________
```

Compiling the model and determining the accuracy

```
inceptionv3_with_aug_model.compile(
                          optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
                          loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
                          metrics=['accuracy'])


inceptionv3_with_aug_history = inceptionv3_with_aug_model.fit(train_generator,
                                            steps_per_epoch=54,
                                            batch_size=32,
                                            validation_data=validation_generator,
                                             validation_steps = 7,
                                             verbose=1,
                                             epochs=EPOCHS)
```

```
Epoch 1/50
54/54 [==============================] - 67s 512ms/step - loss: 0.3437 - accuracy: 0.8646 - val_loss: 0.4023 - val_accuracy: 0.8
Epoch 2/50
54/54 [==============================] - 27s 508ms/step - loss: 0.0933 - accuracy: 0.9675 - val_loss: 0.2271 - val_accuracy: 0.9
Epoch 3/50
54/54 [==============================] - 26s 483ms/step - loss: 0.0565 - accuracy: 0.9826 - val_loss: 0.0632 - val_accuracy: 0.9
Epoch 4/50
54/54 [==============================] - 26s 476ms/step - loss: 0.0425 - accuracy: 0.9831 - val_loss: 0.0337 - val_accuracy: 0.9
Epoch 5/50
54/54 [==============================] - 26s 480ms/step - loss: 0.0212 - accuracy: 0.9959 - val_loss: 0.0322 - val_accuracy: 0.9
```

```
Epoch 6/50
54/54 [==============================] - 26s 476ms/step - loss: 0.0257 - accuracy: 0.9919 - val_loss: 0.0237 - val_accuracy: 0.99
Epoch 7/50
54/54 [==============================] - 26s 473ms/step - loss: 0.0415 - accuracy: 0.9850 - val_loss: 0.0586 - val_accuracy: 0.9
Epoch 8/50
54/54 [==============================] - 26s 475ms/step - loss: 0.0221 - accuracy: 0.9913 - val_loss: 0.0821 - val_accuracy: 0.9
Epoch 9/50
54/54 [==============================] - 26s 476ms/step - loss: 0.0337 - accuracy: 0.9901 - val_loss: 0.0414 - val_accuracy: 0.99
Epoch 10/50
54/54 [==============================] - 26s 475ms/step - loss: 0.0204 - accuracy: 0.9930 - val_loss: 0.0141 - val_accuracy: 0.99
Epoch 11/50
54/54 [==============================] - 26s 472ms/step - loss: 0.0186 - accuracy: 0.9919 - val_loss: 0.0108 - val_accuracy: 0.99
Epoch 12/50
54/54 [==============================] - 26s 472ms/step - loss: 0.0078 - accuracy: 0.9977 - val_loss: 0.0298 - val_accuracy: 0.99
Epoch 13/50
54/54 [==============================] - 26s 479ms/step - loss: 0.0163 - accuracy: 0.9954 - val_loss: 0.0183 - val_accuracy: 0.99
Epoch 14/50
54/54 [==============================] - 26s 477ms/step - loss: 0.0086 - accuracy: 0.9983 - val_loss: 0.0288 - val_accuracy: 0.99
Epoch 15/50
54/54 [==============================] - 25s 469ms/step - loss: 0.0092 - accuracy: 0.9977 - val_loss: 0.0024 - val_accuracy: 1.00
Epoch 16/50
54/54 [==============================] - 26s 475ms/step - loss: 0.0121 - accuracy: 0.9959 - val_loss: 0.0957 - val_accuracy: 0.9
Epoch 17/50
54/54 [==============================] - 25s 470ms/step - loss: 0.0076 - accuracy: 0.9977 - val_loss: 0.0182 - val_accuracy: 0.99
Epoch 18/50
54/54 [==============================] - 26s 472ms/step - loss: 0.0047 - accuracy: 0.9983 - val_loss: 0.0473 - val_accuracy: 0.9
Epoch 19/50
54/54 [==============================] - 25s 464ms/step - loss: 0.0090 - accuracy: 0.9971 - val_loss: 0.0371 - val_accuracy: 0.99
Epoch 20/50
54/54 [==============================] - 25s 470ms/step - loss: 0.0056 - accuracy: 0.9988 - val_loss: 0.0148 - val_accuracy: 0.99
Epoch 21/50
54/54 [==============================] - 25s 471ms/step - loss: 0.0222 - accuracy: 0.9919 - val_loss: 0.0122 - val_accuracy: 0.99
Epoch 22/50
54/54 [==============================] - 26s 477ms/step - loss: 0.0128 - accuracy: 0.9959 - val_loss: 0.0495 - val_accuracy: 0.9
Epoch 23/50
54/54 [==============================] - 26s 481ms/step - loss: 0.0164 - accuracy: 0.9948 - val_loss: 0.1491 - val_accuracy: 0.9
Epoch 24/50
54/54 [==============================] - 26s 478ms/step - loss: 0.0297 - accuracy: 0.9913 - val_loss: 0.0363 - val_accuracy: 0.9
Epoch 25/50
54/54 [==============================] - 26s 477ms/step - loss: 0.0064 - accuracy: 0.9983 - val_loss: 0.0044 - val_accuracy: 1.00
Epoch 26/50
54/54 [==============================] - 26s 477ms/step - loss: 0.0220 - accuracy: 0.9919 - val_loss: 0.0163 - val_accuracy: 0.99
Epoch 27/50
54/54 [==============================] - 26s 475ms/step - loss: 0.0049 - accuracy: 0.9994 - val_loss: 0.0124 - val_accuracy: 0.99
Epoch 28/50
54/54 [==============================] - 26s 477ms/step - loss: 0.0090 - accuracy: 0.9983 - val_loss: 0.0598 - val_accuracy: 0.9
Epoch 29/50
```

Producing the test accuracy and test loss

```
inceptionv3_with_aug_scores = inceptionv3_with_aug_model.evaluate(test_generator)
inceptionv3_with_aug_test_acc = inceptionv3_with_aug_scores[1]*100
inceptionv3_with_aug_test_loss = inceptionv3_with_aug_scores[0]
print("Test Loss: ", inceptionv3_with_aug_test_loss)
print("Test Accuracy: ", inceptionv3_with_aug_test_acc)
```

```
13/13 [==============================] - 5s 411ms/step - loss: 0.0022 - accuracy: 1.0000
Test Loss:  0.002167551079764962
Test Accuracy:  100.0
```

InceptionV3 plot for accuracy vs loss

```
inceptionv3_with_aug_history
```

```
<keras.callbacks.History at 0x7fe290a8e1a0>
```

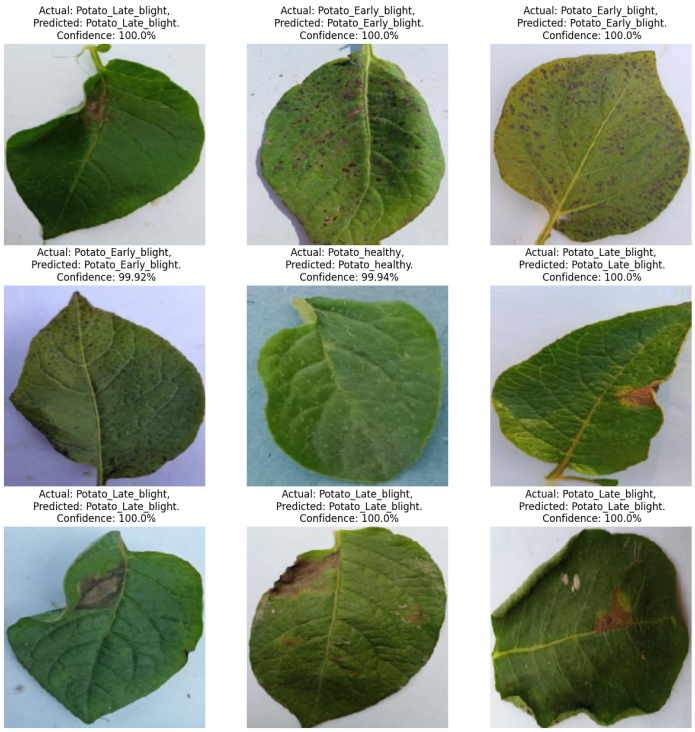
```
inceptionv3_with_aug_acc = inceptionv3_with_aug_history.history['accuracy']
inceptionv3_with_aug_val_acc = inceptionv3_with_aug_history.history['val_accuracy']

inceptionv3_with_aug_loss = inceptionv3_with_aug_history.history['loss']
inceptionv3_with_aug_val_loss = inceptionv3_with_aug_history.history['val_loss']

n = range(1, EPOCHS+1)


plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(n, inceptionv3_with_aug_acc, label='Training Accuracy')
plt.plot(n, inceptionv3_with_aug_val_acc, label='Validation Accuracy')
plt.legend(loc='best')
plt.xlabel('Epochs')
```

```
plt.ylabel('Accuracy')
plt.title('Model Accuracy')

plt.subplot(1,2,2)
plt.plot(n, inceptionv3_with_aug_loss, label='Training Loss')
plt.plot(n, inceptionv3_with_aug_val_loss, label='Validation Loss')
plt.legend(loc='best')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Model Loss')
plt.tight_layout()
plt.suptitle('InceptionV3 Model')
plt.savefig("plot_inceptionv3.png")
plt.show()
```

Prediction of images using InceptionV3

```
def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i])
    img_array = tf.expand_dims(img_array, 0)

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence


plt.figure(figsize=(15, 15))
for images, labels in test_generator:
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i])

        predicted_class, confidence = predict(inceptionv3_with_aug_model, images[i])
        actual_class = class_names[int(labels[i])]

        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n Confidence: {confidence}%")

        plt.axis("off")
    break

plt.savefig("InceptionV3_prediction images")
```

```
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 28ms/step
```



ResNet50

```
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 3

def create_model():
    model = models.Sequential()
    model.add(ResNet50V2(include_top = False, weights = 'imagenet', input_shape = (IMAGE_SIZE, IMAGE_SIZE, CHANNELS)))
    model.add(layers.GlobalAveragePooling2D())
    model.add(layers.Flatten())
    model.add(layers.Dense(256, activation = 'relu'))
    model.add(layers.Dropout(0.3))
    model.add(layers.Dense(n_classes, activation='softmax'))

    return model


resnet50_with_aug_model = create_model()
resnet50_with_aug_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50v2_weights_tf_dim_ordering_tf_ker
94668760/94668760 [==============================] - 3s 0us/step
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50v2 (Functional)     (None, 7, 7, 2048)        23564800

 global_average_pooling2d_2   (None, 2048)             0
 (GlobalAveragePooling2D)

 flatten_2 (Flatten)         (None, 2048)              0
```

```
 dense_4 (Dense)             (None, 256)               524544

 dropout_2 (Dropout)         (None, 256)               0

 dense_5 (Dense)             (None, 3)                 771

=================================================================
Total params: 24,090,115
Trainable params: 24,044,675
Non-trainable params: 45,440
_________________________________________________________________
```

Compiling the model and determining the accuracy

```
resnet50_with_aug_model.compile(
                          optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
                          loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
                          metrics=['accuracy'])


resnet50_with_aug_history = resnet50_with_aug_model.fit(train_generator,
                                                steps_per_epoch=54,
                                                batch_size=32,
                                                validation_data=validation_generator,
                                                 validation_steps = 7,
                                                 verbose=1,
                                                 epochs=EPOCHS)
```

```
Epoch 1/50
54/54 [==============================] - 60s 508ms/step - loss: 0.3488 - accuracy: 0.8542 - val_loss: 0.7331 - val_accuracy: 0.7
Epoch 2/50
54/54 [==============================] - 26s 481ms/step - loss: 0.1123 - accuracy: 0.9659 - val_loss: 0.3463 - val_accuracy: 0.9
Epoch 3/50
54/54 [==============================] - 26s 475ms/step - loss: 0.0531 - accuracy: 0.9797 - val_loss: 0.1987 - val_accuracy: 0.9
Epoch 4/50
54/54 [==============================] - 26s 477ms/step - loss: 0.0498 - accuracy: 0.9884 - val_loss: 0.1177 - val_accuracy: 0.9
Epoch 5/50
54/54 [==============================] - 26s 477ms/step - loss: 0.0377 - accuracy: 0.9890 - val_loss: 0.0510 - val_accuracy: 0.9
Epoch 6/50
54/54 [==============================] - 26s 477ms/step - loss: 0.0287 - accuracy: 0.9907 - val_loss: 0.0318 - val_accuracy: 0.9
Epoch 7/50
54/54 [==============================] - 26s 475ms/step - loss: 0.0435 - accuracy: 0.9878 - val_loss: 0.0498 - val_accuracy: 0.9
Epoch 8/50
54/54 [==============================] - 25s 469ms/step - loss: 0.0322 - accuracy: 0.9924 - val_loss: 0.1058 - val_accuracy: 0.9
Epoch 9/50
54/54 [==============================] - 26s 482ms/step - loss: 0.0144 - accuracy: 0.9948 - val_loss: 0.0892 - val_accuracy: 0.9
Epoch 10/50
54/54 [==============================] - 27s 492ms/step - loss: 0.0268 - accuracy: 0.9936 - val_loss: 0.0769 - val_accuracy: 0.9
Epoch 11/50
54/54 [==============================] - 26s 480ms/step - loss: 0.0126 - accuracy: 0.9948 - val_loss: 0.0465 - val_accuracy: 0.9
Epoch 12/50
54/54 [==============================] - 26s 477ms/step - loss: 0.0277 - accuracy: 0.9924 - val_loss: 0.0387 - val_accuracy: 0.9
Epoch 13/50
54/54 [==============================] - 26s 479ms/step - loss: 0.0402 - accuracy: 0.9855 - val_loss: 0.0379 - val_accuracy: 0.9
Epoch 14/50
54/54 [==============================] - 25s 471ms/step - loss: 0.0118 - accuracy: 0.9965 - val_loss: 0.0826 - val_accuracy: 0.9
Epoch 15/50
54/54 [==============================] - 25s 466ms/step - loss: 0.0565 - accuracy: 0.9837 - val_loss: 0.0611 - val_accuracy: 0.9
Epoch 16/50
54/54 [==============================] - 26s 482ms/step - loss: 0.0350 - accuracy: 0.9890 - val_loss: 0.0598 - val_accuracy: 0.9
Epoch 17/50
54/54 [==============================] - 26s 476ms/step - loss: 0.0165 - accuracy: 0.9959 - val_loss: 0.0290 - val_accuracy: 0.9
Epoch 18/50
54/54 [==============================] - 26s 478ms/step - loss: 0.0173 - accuracy: 0.9954 - val_loss: 0.0292 - val_accuracy: 0.9
Epoch 19/50
54/54 [==============================] - 26s 481ms/step - loss: 0.0226 - accuracy: 0.9942 - val_loss: 0.0266 - val_accuracy: 0.9
Epoch 20/50
54/54 [==============================] - 26s 476ms/step - loss: 0.0214 - accuracy: 0.9924 - val_loss: 0.0272 - val_accuracy: 0.9
Epoch 21/50
54/54 [==============================] - 26s 474ms/step - loss: 0.0184 - accuracy: 0.9936 - val_loss: 0.0494 - val_accuracy: 0.9
Epoch 22/50
54/54 [==============================] - 26s 481ms/step - loss: 0.0281 - accuracy: 0.9907 - val_loss: 0.0705 - val_accuracy: 0.9
Epoch 23/50
54/54 [==============================] - 26s 479ms/step - loss: 0.0257 - accuracy: 0.9924 - val_loss: 0.0559 - val_accuracy: 0.9
Epoch 24/50
54/54 [==============================] - 26s 479ms/step - loss: 0.0207 - accuracy: 0.9942 - val_loss: 0.0488 - val_accuracy: 0.9
Epoch 25/50
54/54 [==============================] - 25s 469ms/step - loss: 0.0268 - accuracy: 0.9948 - val_loss: 0.0273 - val_accuracy: 0.9
Epoch 26/50
54/54 [==============================] - 26s 474ms/step - loss: 0.0109 - accuracy: 0.9965 - val_loss: 0.0188 - val_accuracy: 0.9
Epoch 27/50
54/54 [==============================] - 26s 476ms/step - loss: 0.0042 - accuracy: 0.9988 - val_loss: 0.0079 - val_accuracy: 0.9
Epoch 28/50
54/54 [==============================] - 26s 480ms/step - loss: 0.0016 - accuracy: 1.0000 - val_loss: 0.0057 - val_accuracy: 0.9
Epoch 29/50
```

Producing the test accuracy and test loss

```
resnet50_with_aug_scores = resnet50_with_aug_model.evaluate(test_generator)
resnet50_with_aug_test_acc = resnet50_with_aug_scores[1]*100
resnet50_with_aug_test_loss = resnet50_with_aug_scores[0]
print("Test Loss: ", resnet50_with_aug_test_loss)
print("Test Accuracy: ", resnet50_with_aug_test_acc)
```

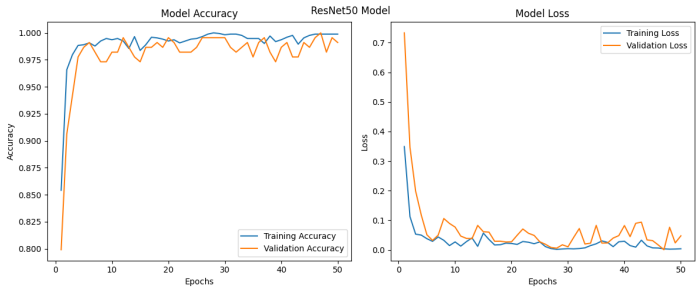
```
13/13 [==============================] - 5s 391ms/step - loss: 0.0040 - accuracy: 0.9976
Test Loss:  0.003967908211052418
Test Accuracy:  99.75550174713135
```

ResNet50 plot for accuracy vs loss

```
resnet50_with_aug_history
```

```
<keras.callbacks.History at 0x7fe2576ad090>
```

```
resnet50_with_aug_acc = resnet50_with_aug_history.history['accuracy']
resnet50_with_aug_val_acc = resnet50_with_aug_history.history['val_accuracy']

resnet50_with_aug_loss = resnet50_with_aug_history.history['loss']
resnet50_with_aug_val_loss = resnet50_with_aug_history.history['val_loss']

n = range(1, EPOCHS+1)


plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(n, resnet50_with_aug_acc, label='Training Accuracy')
plt.plot(n, resnet50_with_aug_val_acc, label='Validation Accuracy')
plt.legend(loc='best')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')

plt.subplot(1,2,2)
plt.plot(n, resnet50_with_aug_loss, label='Training Loss')
plt.plot(n, resnet50_with_aug_val_loss, label='Validation Loss')
plt.legend(loc='best')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Model Loss')
plt.tight_layout()
plt.suptitle('ResNet50 Model')
plt.savefig("plot_resnet50.png")
plt.show()
```



Prediction of images using ResNet50

```
def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i])
    img_array = tf.expand_dims(img_array, 0)

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence


plt.figure(figsize=(15, 15))
for images, labels in test_generator:
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i])

        predicted_class, confidence = predict(resnet50_with_aug_model, images[i])
        actual_class = class_names[int(labels[i])]

        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n Confidence: {confidence}%")

        plt.axis("off")
    break

plt.savefig("ResNet50V2_prediction images")
```

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

Compare the transfer learning models performances

```
1/1 [==============================] - 0s 23ms/step
```

Comparing the test accuracies

```
algorithms = ['VGG19', 'InceptionV3', 'ResNet50']
accuracy = [vgg19_with_aug_test_acc, inceptionv3_with_aug_test_acc, resnet50_with_aug_test_acc]
accuracy = np.around([i for i in accuracy], 3)
colours = ['red', 'green', 'blue']

fig = plt.figure(figsize=(15,15))

# Plotting the bar chart
bars = plt.bar(algorithms, accuracy, color=colours, width = 0.5)

plt.xlabel("Algorithms Used")
plt.ylabel("Testing Accuracy")
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x(), yval+2, yval)

plt.title('Comparison of the test accuracies from the models used')
plt.tight_layout()
plt.savefig('comparison_of_test_accuracies.png')
plt.show()
```

Comparing the test loss

```
algorithms = ['VGG19', 'InceptionV3', 'ResNet50']
loss = [vgg19_with_aug_test_loss, inceptionv3_with_aug_test_loss, resnet50_with_aug_test_loss]
loss = np.around([i for i in loss], 3)
colours = ['red', 'green', 'blue']

fig = plt.figure(figsize=(15,15))

# Plotting the bar chart
bars = plt.bar(algorithms, loss, color=colours, width = 0.5)

plt.xlabel("Algorithms Used")
plt.ylabel("Testing Loss")
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x(), yval+0.00009, yval)

plt.title('Comparison of the test losses from the models used')
plt.tight_layout()
plt.savefig('comparison_of_test_losses.png')
plt.show()
```

```
!jupyter nbconvert --to html PlantVillageTransferLearning.ipynb
```

```
    [NbConvertApp] WARNING | pattern 'PlantVillageTransferLearning.ipynb' matched no files
    This application is used to convert notebook files (*.ipynb)
```

```
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and include the error message in the cell output (the d
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_format=notebook --FilesWriter.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_format=notebook --FilesWriter.build_directory=
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True --TemplateExporter.exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True --TemplateExporter.exclude_input=True --TemplateExporter.exclu
--allow-chromium-download
    Whether to allow downloading chromium if no suitable version is found on the system.
```