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)
```

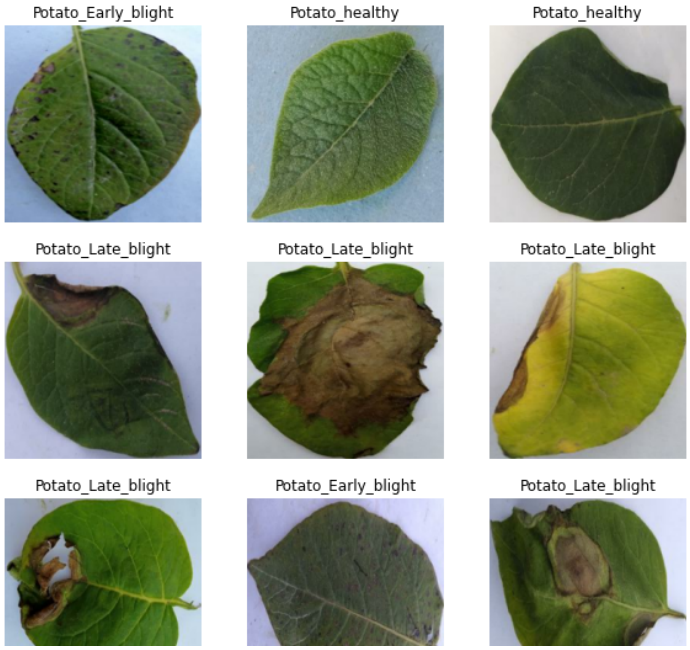
```
{'Potato_Early_blight': 1628, 'Potato_healthy': 1020, 'Potato_Late_blight': 1424}
```

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

```
(32, 224, 224, 3)
[1 2 0 1 0 0 0 0 2 0 1 2 0 0 2 1 0 0 1 1 2 1 2 0 2 0 1 2 1 2 0 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 [00:38, 106.94 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 [==============================] - 0s 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= 'adam',
                          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 [==============================] - 33s 591ms/step - loss: 1.0798 - accuracy: 0.3998 - val_loss: 1.0772 - val_accuracy: 0.4
Epoch 2/50
54/54 [==============================] - 31s 577ms/step - loss: 1.0763 - accuracy: 0.4143 - val_loss: 1.0821 - val_accuracy: 0.3
Epoch 3/50
54/54 [==============================] - 32s 586ms/step - loss: 1.0790 - accuracy: 0.4062 - val_loss: 1.0813 - val_accuracy: 0.4
Epoch 4/50
54/54 [==============================] - 32s 580ms/step - loss: 1.0799 - accuracy: 0.4073 - val_loss: 1.0887 - val_accuracy: 0.4
Epoch 5/50
54/54 [==============================] - 32s 592ms/step - loss: 1.0821 - accuracy: 0.3957 - val_loss: 1.0761 - val_accuracy: 0.4
Epoch 6/50
54/54 [==============================] - 32s 581ms/step - loss: 1.0835 - accuracy: 0.3974 - val_loss: 1.0952 - val_accuracy: 0.3
Epoch 7/50
54/54 [==============================] - 32s 596ms/step - loss: 1.0722 - accuracy: 0.4079 - val_loss: 1.0634 - val_accuracy: 0.4
Epoch 8/50
54/54 [==============================] - 31s 575ms/step - loss: 1.0741 - accuracy: 0.4172 - val_loss: 1.0994 - val_accuracy: 0.3
Epoch 9/50
54/54 [==============================] - 31s 574ms/step - loss: 1.0823 - accuracy: 0.4114 - val_loss: 1.0818 - val_accuracy: 0.4
Epoch 10/50
54/54 [==============================] - 31s 576ms/step - loss: 1.0852 - accuracy: 0.3911 - val_loss: 1.0819 - val_accuracy: 0.4
Epoch 11/50
54/54 [==============================] - 31s 573ms/step - loss: 1.0797 - accuracy: 0.3998 - val_loss: 1.0812 - val_accuracy: 0.4
Epoch 12/50
54/54 [==============================] - 32s 592ms/step - loss: 1.0801 - accuracy: 0.4080 - val_loss: 1.0838 - val_accuracy: 0.3
Epoch 13/50
54/54 [==============================] - 32s 578ms/step - loss: 1.0801 - accuracy: 0.3969 - val_loss: 1.0893 - val_accuracy: 0.4
Epoch 14/50
54/54 [==============================] - 32s 595ms/step - loss: 1.0821 - accuracy: 0.3964 - val_loss: 1.0804 - val_accuracy: 0.4
Epoch 15/50
54/54 [==============================] - 32s 579ms/step - loss: 1.0779 - accuracy: 0.4022 - val_loss: 1.0771 - val_accuracy: 0.4
Epoch 16/50
54/54 [==============================] - 32s 582ms/step - loss: 1.0815 - accuracy: 0.4057 - val_loss: 1.0727 - val_accuracy: 0.4
Epoch 17/50
54/54 [==============================] - 32s 580ms/step - loss: 1.0843 - accuracy: 0.3899 - val_loss: 1.0980 - val_accuracy: 0.3
Epoch 18/50
54/54 [==============================] - 31s 578ms/step - loss: 1.0831 - accuracy: 0.3928 - val_loss: 1.0860 - val_accuracy: 0.3
Epoch 19/50
54/54 [==============================] - 31s 576ms/step - loss: 1.0794 - accuracy: 0.4003 - val_loss: 1.0971 - val_accuracy: 0.3
Epoch 20/50
54/54 [==============================] - 32s 580ms/step - loss: 1.0825 - accuracy: 0.3935 - val_loss: 1.0804 - val_accuracy: 0.3
Epoch 21/50
54/54 [==============================] - 32s 597ms/step - loss: 1.0829 - accuracy: 0.3964 - val_loss: 1.0833 - val_accuracy: 0.4
Epoch 22/50
54/54 [==============================] - 31s 574ms/step - loss: 1.0861 - accuracy: 0.4021 - val_loss: 1.0961 - val_accuracy: 0.3
Epoch 23/50
54/54 [==============================] - 31s 577ms/step - loss: 1.0824 - accuracy: 0.3935 - val_loss: 1.0850 - val_accuracy: 0.3
Epoch 24/50
54/54 [==============================] - 31s 575ms/step - loss: 1.0804 - accuracy: 0.3895 - val_loss: 1.0750 - val_accuracy: 0.3
Epoch 25/50
54/54 [==============================] - 31s 573ms/step - loss: 1.0785 - accuracy: 0.4027 - val_loss: 1.0746 - val_accuracy: 0.4
Epoch 26/50
54/54 [==============================] - 31s 572ms/step - loss: 1.0831 - accuracy: 0.3963 - val_loss: 1.0970 - val_accuracy: 0.3
Epoch 27/50
54/54 [==============================] - 31s 574ms/step - loss: 1.0788 - accuracy: 0.4091 - val_loss: 1.0847 - val_accuracy: 0.3
Epoch 28/50
54/54 [==============================] - 33s 599ms/step - loss: 1.0836 - accuracy: 0.4028 - val_loss: 1.0863 - val_accuracy: 0.3
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)
```

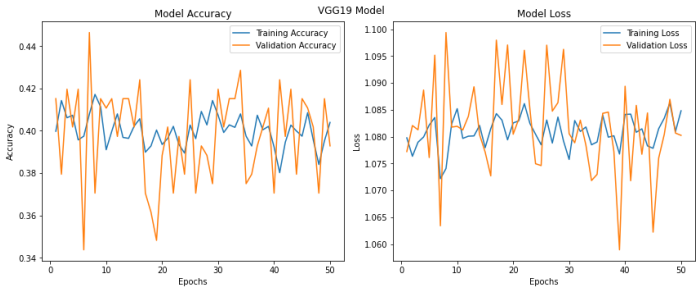
```
13/13 [==============================] - 6s 416ms/step - loss: 1.0803 - accuracy: 0.4010
Test Loss:  1.080260157585144
Test Accuracy:  40.09779989719391
```

VGG19 plot for accuracy vs loss

```
vgg19_with_aug_history
```

```
<keras.callbacks.History at 0x7fc97c4cde50>
```

```
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()
```

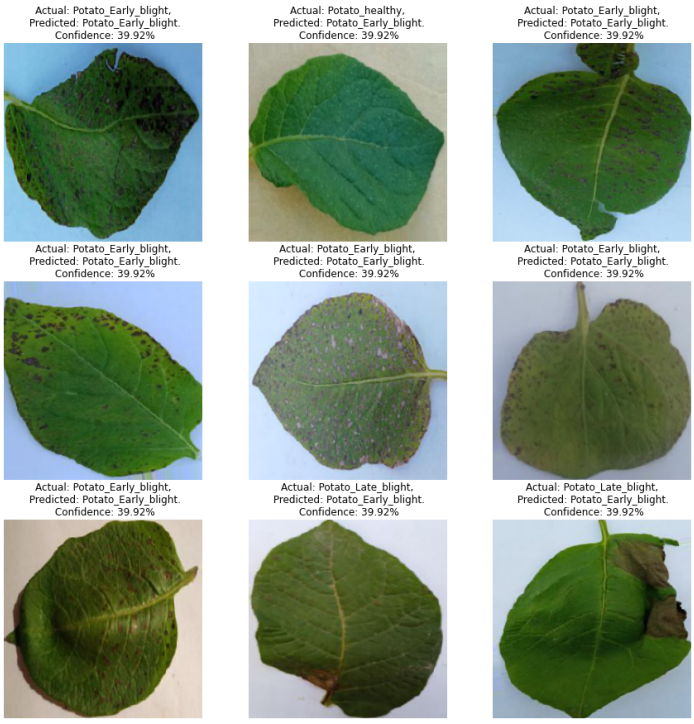


Prediction of images using VGG19

```
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 139ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 15ms/step
```



GoogLeNet (InceptionV3)

```
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


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

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 7, 7, 512)         20024384

 global_average_pooling2d_3   (None, 512)              0
 (GlobalAveragePooling2D)

 flatten_3 (Flatten)         (None, 512)               0

 dense_6 (Dense)             (None, 256)               131328

 dropout_3 (Dropout)         (None, 256)               0

 dense_7 (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

```
inceptionv3_with_aug_model.compile(
                          optimizer='adam',
                          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 [==============================] - 35s 624ms/step - loss: 1.5829 - accuracy: 0.3847 - val_loss: 1.0872 - val_accuracy: 0.3
Epoch 2/50
54/54 [==============================] - 31s 576ms/step - loss: 1.0894 - accuracy: 0.3715 - val_loss: 1.0824 - val_accuracy: 0.4
Epoch 3/50
54/54 [==============================] - 32s 577ms/step - loss: 1.0920 - accuracy: 0.3951 - val_loss: 1.0758 - val_accuracy: 0.4
Epoch 4/50
54/54 [==============================] - 33s 611ms/step - loss: 1.0848 - accuracy: 0.3916 - val_loss: 1.0816 - val_accuracy: 0.4
Epoch 5/50
54/54 [==============================] - 32s 583ms/step - loss: 1.0840 - accuracy: 0.3916 - val_loss: 1.0808 - val_accuracy: 0.4
Epoch 6/50
54/54 [==============================] - 32s 578ms/step - loss: 1.0824 - accuracy: 0.4034 - val_loss: 1.0853 - val_accuracy: 0.3
Epoch 7/50
54/54 [==============================] - 32s 578ms/step - loss: 1.0800 - accuracy: 0.4120 - val_loss: 1.0865 - val_accuracy: 0.3
Epoch 8/50
54/54 [==============================] - 32s 583ms/step - loss: 1.0831 - accuracy: 0.3900 - val_loss: 1.0663 - val_accuracy: 0.4
Epoch 9/50
54/54 [==============================] - 33s 598ms/step - loss: 1.0861 - accuracy: 0.3993 - val_loss: 1.0788 - val_accuracy: 0.3
Epoch 10/50
54/54 [==============================] - 32s 582ms/step - loss: 1.0820 - accuracy: 0.3957 - val_loss: 1.0719 - val_accuracy: 0.4
Epoch 11/50
54/54 [==============================] - 32s 578ms/step - loss: 1.0829 - accuracy: 0.3928 - val_loss: 1.0877 - val_accuracy: 0.3
Epoch 12/50
54/54 [==============================] - 32s 580ms/step - loss: 1.0843 - accuracy: 0.3941 - val_loss: 1.0873 - val_accuracy: 0.3
Epoch 13/50
54/54 [==============================] - 31s 575ms/step - loss: 1.0774 - accuracy: 0.4085 - val_loss: 1.0760 - val_accuracy: 0.4
Epoch 14/50
54/54 [==============================] - 31s 577ms/step - loss: 1.0787 - accuracy: 0.4086 - val_loss: 1.0854 - val_accuracy: 0.3
Epoch 15/50
54/54 [==============================] - 32s 579ms/step - loss: 1.0872 - accuracy: 0.3791 - val_loss: 1.0852 - val_accuracy: 0.3
Epoch 16/50
54/54 [==============================] - 31s 576ms/step - loss: 1.0808 - accuracy: 0.4015 - val_loss: 1.0839 - val_accuracy: 0.3
```

```
Epoch 17/50
54/54 [==============================] - 33s 600ms/step - loss: 1.0840 - accuracy: 0.3957 - val_loss: 1.0746 - val_accuracy: 0.4
Epoch 18/50
54/54 [==============================] - 32s 581ms/step - loss: 1.0846 - accuracy: 0.3831 - val_loss: 1.0734 - val_accuracy: 0.3
Epoch 19/50
54/54 [==============================] - 32s 579ms/step - loss: 1.0801 - accuracy: 0.4074 - val_loss: 1.0721 - val_accuracy: 0.4
Epoch 20/50
54/54 [==============================] - 31s 578ms/step - loss: 1.0724 - accuracy: 0.4259 - val_loss: 1.0838 - val_accuracy: 0.3
Epoch 21/50
54/54 [==============================] - 31s 577ms/step - loss: 1.0817 - accuracy: 0.3998 - val_loss: 1.0912 - val_accuracy: 0.3
Epoch 22/50
54/54 [==============================] - 31s 574ms/step - loss: 1.0808 - accuracy: 0.4132 - val_loss: 1.0847 - val_accuracy: 0.4
Epoch 23/50
54/54 [==============================] - 32s 579ms/step - loss: 1.0860 - accuracy: 0.3980 - val_loss: 1.0957 - val_accuracy: 0.3
Epoch 24/50
54/54 [==============================] - 32s 579ms/step - loss: 1.0778 - accuracy: 0.4155 - val_loss: 1.0784 - val_accuracy: 0.3
Epoch 25/50
54/54 [==============================] - 33s 597ms/step - loss: 1.0849 - accuracy: 0.3837 - val_loss: 1.0916 - val_accuracy: 0.3
Epoch 26/50
54/54 [==============================] - 31s 578ms/step - loss: 1.0773 - accuracy: 0.3981 - val_loss: 1.0907 - val_accuracy: 0.3
Epoch 27/50
54/54 [==============================] - 31s 575ms/step - loss: 1.0889 - accuracy: 0.3823 - val_loss: 1.0788 - val_accuracy: 0.4
Epoch 28/50
```

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 [==============================] - 6s 427ms/step - loss: 1.0803 - accuracy: 0.4010
Test Loss:  1.080295443534851
Test Accuracy:  40.09779989719391
```

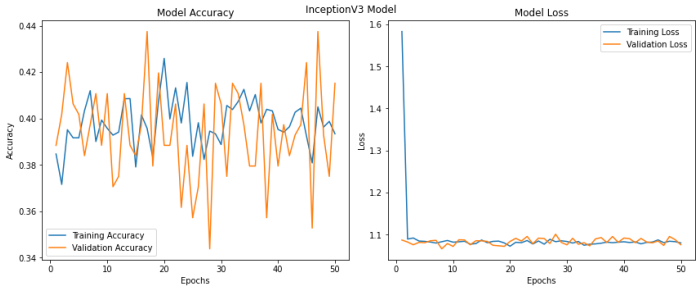
InceptionV3 plot for accuracy vs loss

```
inceptionv3_with_aug_history
```

```
<keras.callbacks.History at 0x7fc97620acd0>
```

```
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)
```

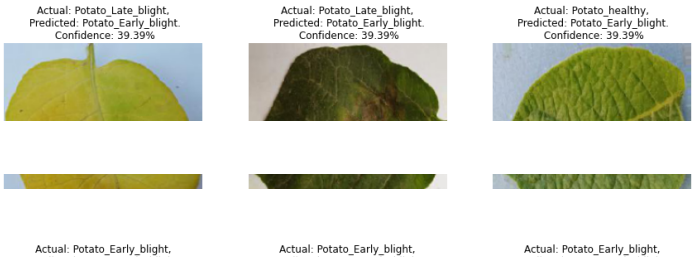
```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(n, inceptionv3_with_aug_acc, label='Training Accuracy')
plt.plot(n, inceptionv3_with_aug_val_acc, label='Validation Accuracy')
plt.legend(loc='best')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')

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 [==============================] - 0s 145ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 15ms/step
```



ResNet50

```
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
```

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

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 7, 7, 512)         20024384

 global_average_pooling2d_4   (None, 512)              0
 (GlobalAveragePooling2D)

 flatten_4 (Flatten)         (None, 512)               0

 dense_8 (Dense)             (None, 256)               131328

 dropout_4 (Dropout)         (None, 256)               0

 dense_9 (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

```
resnet50_with_aug_model.compile(
                          optimizer='adam',
                          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 [==============================] - 33s 593ms/step - loss: 1.5126 - accuracy: 0.3736 - val_loss: 1.0890 - val_accuracy: 0.3
Epoch 2/50
```

```
54/54 [==============================] - 32s 585ms/step - loss: 1.0846 - accuracy: 0.3999 - val_loss: 1.0897 - val_accuracy: 0.3
Epoch 3/50
54/54 [==============================] - 32s 584ms/step - loss: 1.0908 - accuracy: 0.4033 - val_loss: 1.0782 - val_accuracy: 0.4
Epoch 4/50
54/54 [==============================] - 32s 585ms/step - loss: 1.0812 - accuracy: 0.4073 - val_loss: 1.0938 - val_accuracy: 0.3
Epoch 5/50
54/54 [==============================] - 32s 584ms/step - loss: 1.0794 - accuracy: 0.4073 - val_loss: 1.0891 - val_accuracy: 0.3
Epoch 6/50
54/54 [==============================] - 32s 580ms/step - loss: 1.0820 - accuracy: 0.3953 - val_loss: 1.0645 - val_accuracy: 0.4
Epoch 7/50
54/54 [==============================] - 32s 579ms/step - loss: 1.0827 - accuracy: 0.3835 - val_loss: 1.0834 - val_accuracy: 0.4
Epoch 8/50
54/54 [==============================] - 33s 600ms/step - loss: 1.0821 - accuracy: 0.3940 - val_loss: 1.0685 - val_accuracy: 0.4
Epoch 9/50
54/54 [==============================] - 31s 573ms/step - loss: 1.0833 - accuracy: 0.3941 - val_loss: 1.0857 - val_accuracy: 0.3
Epoch 10/50
54/54 [==============================] - 31s 571ms/step - loss: 1.0809 - accuracy: 0.4003 - val_loss: 1.0792 - val_accuracy: 0.4
Epoch 11/50
54/54 [==============================] - 31s 573ms/step - loss: 1.0885 - accuracy: 0.3848 - val_loss: 1.0881 - val_accuracy: 0.3
Epoch 12/50
54/54 [==============================] - 32s 585ms/step - loss: 1.0771 - accuracy: 0.4016 - val_loss: 1.0713 - val_accuracy: 0.3
Epoch 13/50
54/54 [==============================] - 32s 577ms/step - loss: 1.0788 - accuracy: 0.4022 - val_loss: 1.0991 - val_accuracy: 0.3
Epoch 14/50
54/54 [==============================] - 31s 577ms/step - loss: 1.0745 - accuracy: 0.4039 - val_loss: 1.0625 - val_accuracy: 0.4
Epoch 15/50
54/54 [==============================] - 32s 579ms/step - loss: 1.0824 - accuracy: 0.4027 - val_loss: 1.0804 - val_accuracy: 0.3
Epoch 16/50
54/54 [==============================] - 33s 597ms/step - loss: 1.0794 - accuracy: 0.4067 - val_loss: 1.0773 - val_accuracy: 0.4
Epoch 17/50
54/54 [==============================] - 32s 578ms/step - loss: 1.0783 - accuracy: 0.4102 - val_loss: 1.0824 - val_accuracy: 0.4
Epoch 18/50
54/54 [==============================] - 33s 596ms/step - loss: 1.0834 - accuracy: 0.4051 - val_loss: 1.0780 - val_accuracy: 0.4
Epoch 19/50
54/54 [==============================] - 33s 606ms/step - loss: 1.0819 - accuracy: 0.3964 - val_loss: 1.0913 - val_accuracy: 0.3
Epoch 20/50
54/54 [==============================] - 33s 605ms/step - loss: 1.0876 - accuracy: 0.3895 - val_loss: 1.0760 - val_accuracy: 0.4
Epoch 21/50
54/54 [==============================] - 33s 599ms/step - loss: 1.0861 - accuracy: 0.3893 - val_loss: 1.0870 - val_accuracy: 0.3
Epoch 22/50
54/54 [==============================] - 32s 584ms/step - loss: 1.0801 - accuracy: 0.4120 - val_loss: 1.0779 - val_accuracy: 0.4
Epoch 23/50
54/54 [==============================] - 32s 586ms/step - loss: 1.0821 - accuracy: 0.4022 - val_loss: 1.0840 - val_accuracy: 0.4
Epoch 24/50
54/54 [==============================] - 32s 586ms/step - loss: 1.0845 - accuracy: 0.3906 - val_loss: 1.0804 - val_accuracy: 0.4
Epoch 25/50
54/54 [==============================] - 33s 604ms/step - loss: 1.0806 - accuracy: 0.4034 - val_loss: 1.0913 - val_accuracy: 0.3
Epoch 26/50
54/54 [==============================] - 32s 580ms/step - loss: 1.0828 - accuracy: 0.3953 - val_loss: 1.1017 - val_accuracy: 0.3
Epoch 27/50
54/54 [==============================] - 32s 581ms/step - loss: 1.0789 - accuracy: 0.3934 - val_loss: 1.0867 - val_accuracy: 0.3
Epoch 28/50
```

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 [==============================] - 6s 434ms/step - loss: 1.0803 - accuracy: 0.4010
Test Loss:  1.0802948474884033
Test Accuracy:  40.09779989719391
```

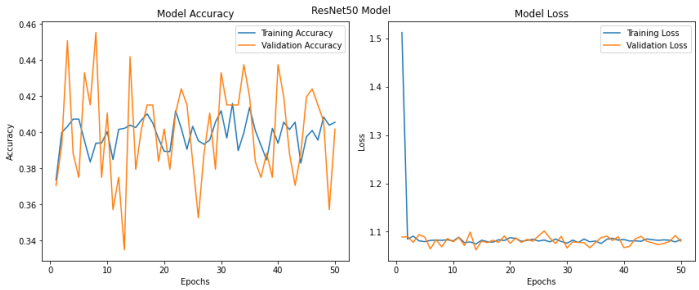
ResNet50 plot for accuracy vs loss

```
resnet50_with_aug_history
```

```
<keras.callbacks.History at 0x7fc8e64234c0>
```

```
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)
```

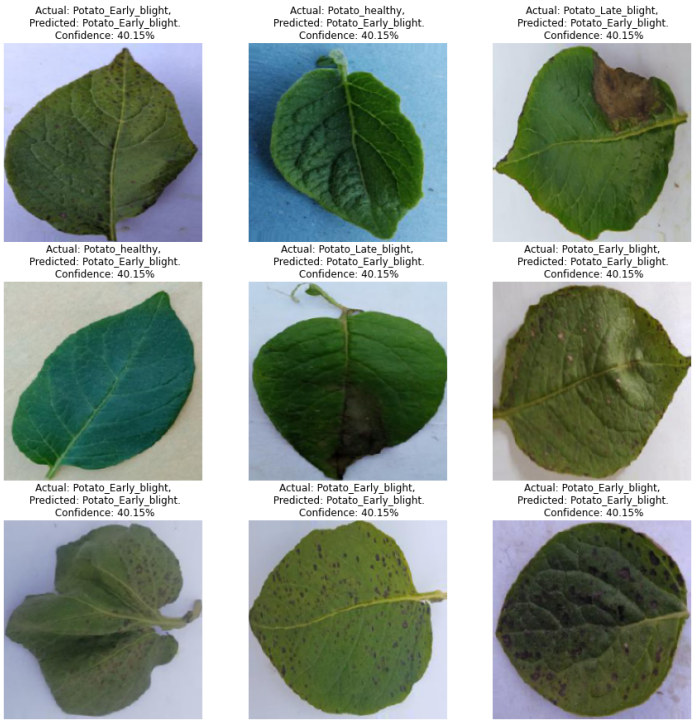
```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(n, resnet50_with_aug_acc, label='Training Accuracy')
plt.plot(n, resnet50_with_aug_val_acc, label='Validation Accuracy')
plt.legend(loc='best')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')

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")
```

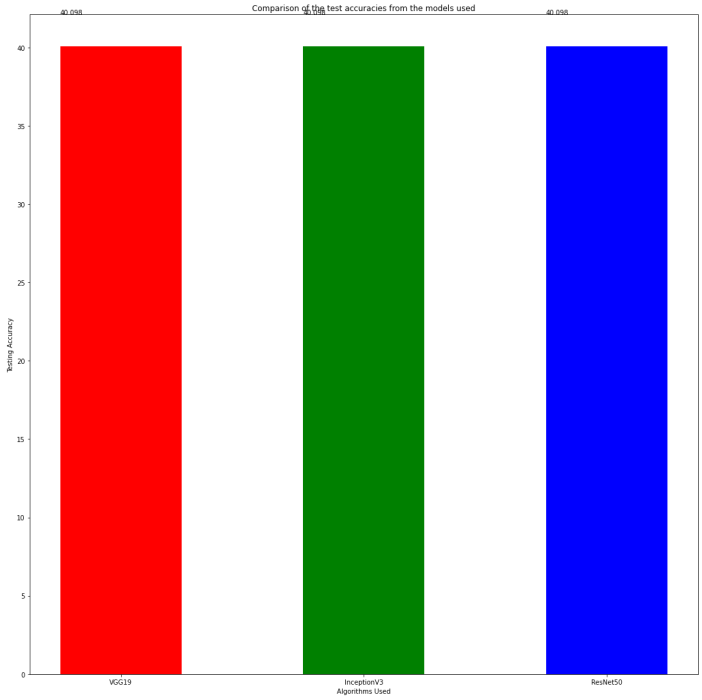
```
1/1 [==============================] - 0s 188ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 17ms/step
```



Compare the transfer learning models performances

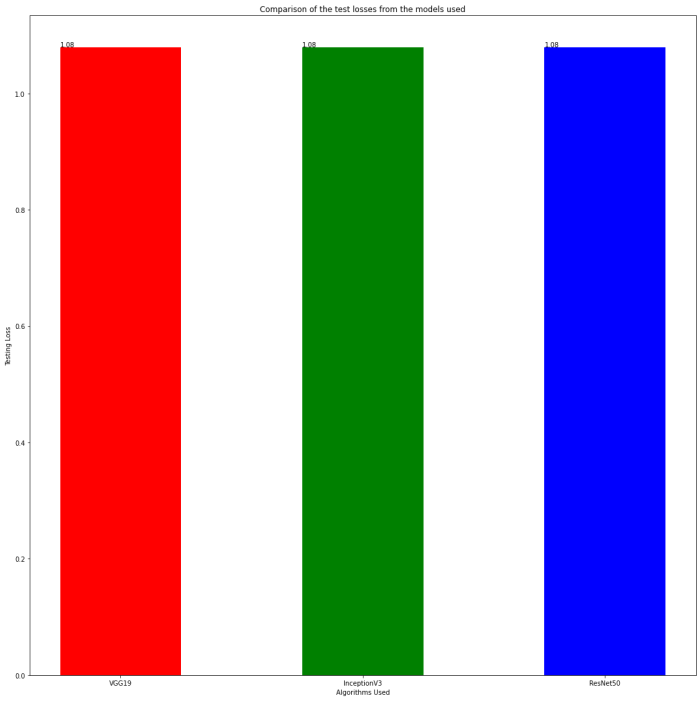
Comparing the test accuracies

```
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] Converting notebook PlantVillageTransferLearning.ipynb to html
[NbConvertApp] Writing 5359748 bytes to PlantVillageTransferLearning.html
```