Mini Project EC 9170 Plant disease detection 2020/E/014 2020/E/082 2020/E/085 Group 15

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

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

```
import os
import shutil
from sklearn.model_selection import train_test_split


dataset_path = '/content/drive/MyDrive/plant_disease_detection'
diseased_path = os.path.join(dataset_path, 'diseased')
healthy_path = os.path.join(dataset_path, 'healthy')



base_dir = '/content/drive/MyDrive/plant_disease_detection'
train_dir = os.path.join(base_dir, 'train')
val_dir = os.path.join(base_dir, 'val')
test_dir = os.path.join(base_dir, 'test')

for dir in [train_dir, val_dir, test_dir]:
    os.makedirs(os.path.join(dir, 'diseased'), exist_ok=True)
    os.makedirs(os.path.join(dir, 'healthy'), exist_ok=True)


# Function to split and move files
def split_and_move_files(src_dir, train_dir, val_dir, test_dir, class_name):
    files = os.listdir(src_dir)
    train_files, test_files = train_test_split(files, test_size=0.1, random_state=42)
    train_files, val_files = train_test_split(train_files, test_size=0.2, random_state=42)

    for file in train_files:
        shutil.move(os.path.join(src_dir, file), os.path.join(train_dir, class_name, file))
    for file in val_files:
        shutil.move(os.path.join(src_dir, file), os.path.join(val_dir, class_name, file))
    for file in test_files:
        shutil.move(os.path.join(src_dir, file), os.path.join(test_dir, class_name, file))
```

split and move images to train,valisation and test folders

```
split_and_move_files(diseased_path, train_dir, val_dir, test_dir, 'diseased')
split_and_move_files(healthy_path, train_dir, val_dir, test_dir, 'healthy')


import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
```

First model - VGG16

```
#VGG16
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

for layer in base_model.layers:
    layer.trainable = False

model = Sequential([
    base_model,
    Flatten(),
    Dense(256, activation='relu'),
    Dense(1, activation='sigmoid')
])

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

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop
58889256/58889256 [==============================] - 0s 0us/step
```

```
#data generatos - VGG16
train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

train_generator = train_datagen.flow_from_directory(
    '/content/drive/MyDrive/plant_disease_detection/train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    '/content/drive/MyDrive/plant_disease_detection/val',
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    subset='validation'
)
```

```
Found 236 images belonging to 2 classes.
Found 14 images belonging to 2 classes.
```

```
# Train the model
history_vgg16 = model.fit(train_generator, epochs=20, validation_data=validation_generator)
```

```
Epoch 1/20
8/8 [==============================] - 142s 18s/step - loss: 0.4876 - accuracy: 0.7797 - val_loss: 0.2595 - val_accuracy: 0.9286
Epoch 2/20
8/8 [==============================] - 139s 17s/step - loss: 0.1007 - accuracy: 0.9746 - val_loss: 0.9589 - val_accuracy: 0.4286
Epoch 3/20
8/8 [==============================] - 141s 18s/step - loss: 0.0657 - accuracy: 0.9831 - val_loss: 1.0709 - val_accuracy: 0.4286
Epoch 4/20
8/8 [==============================] - 139s 17s/step - loss: 0.0387 - accuracy: 0.9915 - val_loss: 0.9801 - val_accuracy: 0.5000
Epoch 5/20
8/8 [==============================] - 139s 17s/step - loss: 0.0258 - accuracy: 0.9958 - val_loss: 1.2705 - val_accuracy: 0.3571
Epoch 6/20
8/8 [==============================] - 141s 18s/step - loss: 0.0187 - accuracy: 0.9958 - val_loss: 1.4131 - val_accuracy: 0.3571
Epoch 7/20
8/8 [==============================] - 139s 17s/step - loss: 0.0135 - accuracy: 1.0000 - val_loss: 1.1732 - val_accuracy: 0.5000
Epoch 8/20
8/8 [==============================] - 138s 17s/step - loss: 0.0101 - accuracy: 1.0000 - val_loss: 1.3087 - val_accuracy: 0.3571
Epoch 9/20
8/8 [==============================] - 139s 17s/step - loss: 0.0078 - accuracy: 1.0000 - val_loss: 1.3774 - val_accuracy: 0.3571
Epoch 10/20
8/8 [==============================] - 139s 17s/step - loss: 0.0068 - accuracy: 1.0000 - val_loss: 1.3381 - val_accuracy: 0.3571
Epoch 11/20
8/8 [==============================] - 140s 17s/step - loss: 0.0055 - accuracy: 1.0000 - val_loss: 1.6184 - val_accuracy: 0.3571
Epoch 12/20
8/8 [==============================] - 140s 18s/step - loss: 0.0052 - accuracy: 1.0000 - val_loss: 1.6439 - val_accuracy: 0.3571
Epoch 13/20
8/8 [==============================] - 140s 17s/step - loss: 0.0044 - accuracy: 1.0000 - val_loss: 1.2823 - val_accuracy: 0.4286
Epoch 14/20
8/8 [==============================] - 139s 17s/step - loss: 0.0039 - accuracy: 1.0000 - val_loss: 1.3502 - val_accuracy: 0.3571
Epoch 15/20
8/8 [==============================] - 139s 17s/step - loss: 0.0034 - accuracy: 1.0000 - val_loss: 1.6170 - val_accuracy: 0.3571
Epoch 16/20
8/8 [==============================] - 143s 18s/step - loss: 0.0030 - accuracy: 1.0000 - val_loss: 1.5918 - val_accuracy: 0.3571
Epoch 17/20
8/8 [==============================] - 140s 17s/step - loss: 0.0027 - accuracy: 1.0000 - val_loss: 1.5414 - val_accuracy: 0.3571
Epoch 18/20
8/8 [==============================] - 143s 18s/step - loss: 0.0024 - accuracy: 1.0000 - val_loss: 1.5761 - val_accuracy: 0.3571
Epoch 19/20
8/8 [==============================] - 142s 18s/step - loss: 0.0022 - accuracy: 1.0000 - val_loss: 1.6158 - val_accuracy: 0.3571
Epoch 20/20
8/8 [==============================] - 140s 17s/step - loss: 0.0021 - accuracy: 1.0000 - val_loss: 1.6861 - val_accuracy: 0.3571
```

```
#save model - VGG16
model.save('/content/drive/MyDrive/plant_disease_detection/vgg16_model.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `
  saving_api.save_model(
```

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
```

Second model - ResNet50

```
#ResNet50
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

for layer in base_model.layers:
    layer.trainable = False

model = Sequential([
    base_model,
    Flatten(),
    Dense(256, activation='relu'),
    Dense(1, activation='sigmoid')
])

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

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_n
94765736/94765736 [==============================] - 1s 0us/step
```

```
#Data generators - ResNet50
train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

train_generator = train_datagen.flow_from_directory(
    '/content/drive/MyDrive/plant_disease_detection/train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    '/content/drive/MyDrive/plant_disease_detection/val',
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    subset='validation'
)
```

```
Found 236 images belonging to 2 classes.
Found 14 images belonging to 2 classes.
```

```
#Train the model - Resnet50
history_resnet50 = model.fit(train_generator, epochs=20, validation_data=validation_generator)
```

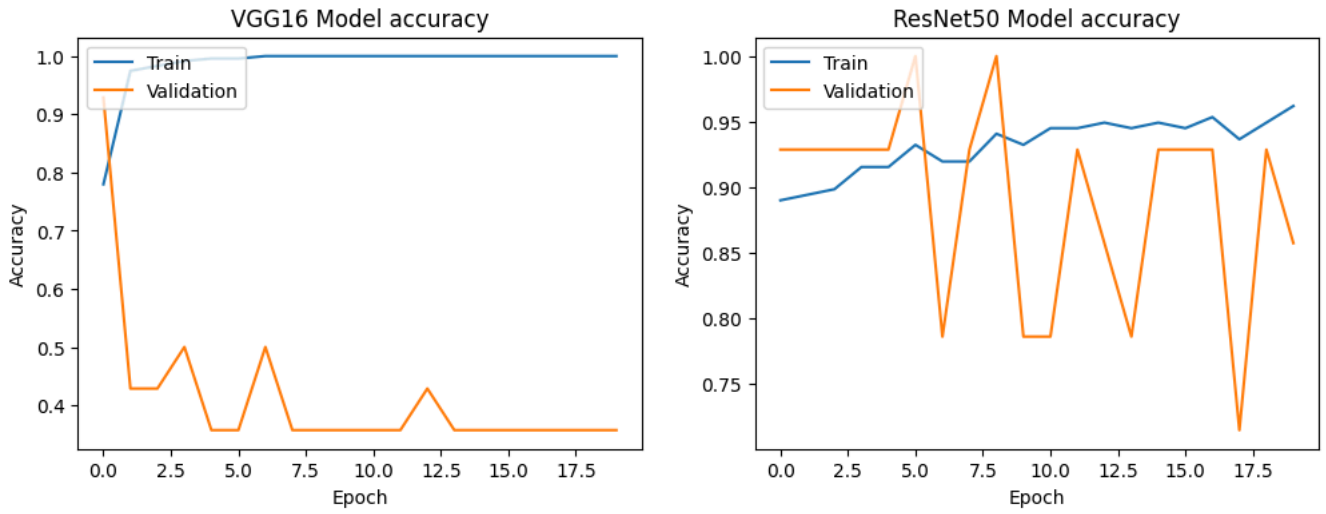
```
Epoch 1/20
8/8 [==============================] - 56s 6s/step - loss: 1.1557 - accuracy: 0.8898 - val_loss: 0.2902 - val_accuracy: 0.9286
Epoch 2/20
8/8 [==============================] - 48s 6s/step - loss: 0.3674 - accuracy: 0.8941 - val_loss: 0.2452 - val_accuracy: 0.9286
Epoch 3/20
8/8 [==============================] - 51s 6s/step - loss: 0.3250 - accuracy: 0.8983 - val_loss: 0.4209 - val_accuracy: 0.9286
Epoch 4/20
8/8 [==============================] - 49s 6s/step - loss: 0.2690 - accuracy: 0.9153 - val_loss: 0.1984 - val_accuracy: 0.9286
Epoch 5/20
8/8 [==============================] - 48s 6s/step - loss: 0.2144 - accuracy: 0.9153 - val_loss: 0.3182 - val_accuracy: 0.9286
Epoch 6/20
8/8 [==============================] - 52s 6s/step - loss: 0.1987 - accuracy: 0.9322 - val_loss: 0.2101 - val_accuracy: 1.0000
Epoch 7/20
8/8 [==============================] - 49s 6s/step - loss: 0.1910 - accuracy: 0.9195 - val_loss: 0.3929 - val_accuracy: 0.7857
Epoch 8/20
8/8 [==============================] - 50s 6s/step - loss: 0.1887 - accuracy: 0.9195 - val_loss: 0.2596 - val_accuracy: 0.9286
Epoch 9/20
8/8 [==============================] - 51s 6s/step - loss: 0.1678 - accuracy: 0.9407 - val_loss: 0.1857 - val_accuracy: 1.0000
Epoch 10/20
8/8 [==============================] - 49s 7s/step - loss: 0.1834 - accuracy: 0.9322 - val_loss: 0.3296 - val_accuracy: 0.7857
Epoch 11/20
8/8 [==============================] - 51s 6s/step - loss: 0.1592 - accuracy: 0.9449 - val_loss: 0.3368 - val_accuracy: 0.7857
```

```
Epoch 12/20
8/8 [==============================] - 49s 6s/step - loss: 0.1523 - accuracy: 0.9449 - val_loss: 0.2122 - val_accuracy: 0.9286
Epoch 13/20
8/8 [==============================] - 50s 6s/step - loss: 0.1450 - accuracy: 0.9492 - val_loss: 0.3131 - val_accuracy: 0.8571
Epoch 14/20
8/8 [==============================] - 52s 6s/step - loss: 0.1362 - accuracy: 0.9449 - val_loss: 0.3262 - val_accuracy: 0.7857
Epoch 15/20
8/8 [==============================] - 49s 6s/step - loss: 0.1490 - accuracy: 0.9492 - val_loss: 0.1923 - val_accuracy: 0.9286
Epoch 16/20
8/8 [==============================] - 48s 6s/step - loss: 0.1306 - accuracy: 0.9449 - val_loss: 0.2578 - val_accuracy: 0.9286
Epoch 17/20
8/8 [==============================] - 49s 6s/step - loss: 0.1242 - accuracy: 0.9534 - val_loss: 0.1776 - val_accuracy: 0.9286
Epoch 18/20
8/8 [==============================] - 54s 7s/step - loss: 0.1279 - accuracy: 0.9364 - val_loss: 0.5170 - val_accuracy: 0.7143
Epoch 19/20
8/8 [==============================] - 49s 6s/step - loss: 0.1239 - accuracy: 0.9492 - val_loss: 0.2319 - val_accuracy: 0.9286
Epoch 20/20
8/8 [==============================] - 51s 6s/step - loss: 0.1080 - accuracy: 0.9619 - val_loss: 0.2636 - val_accuracy: 0.8571
```

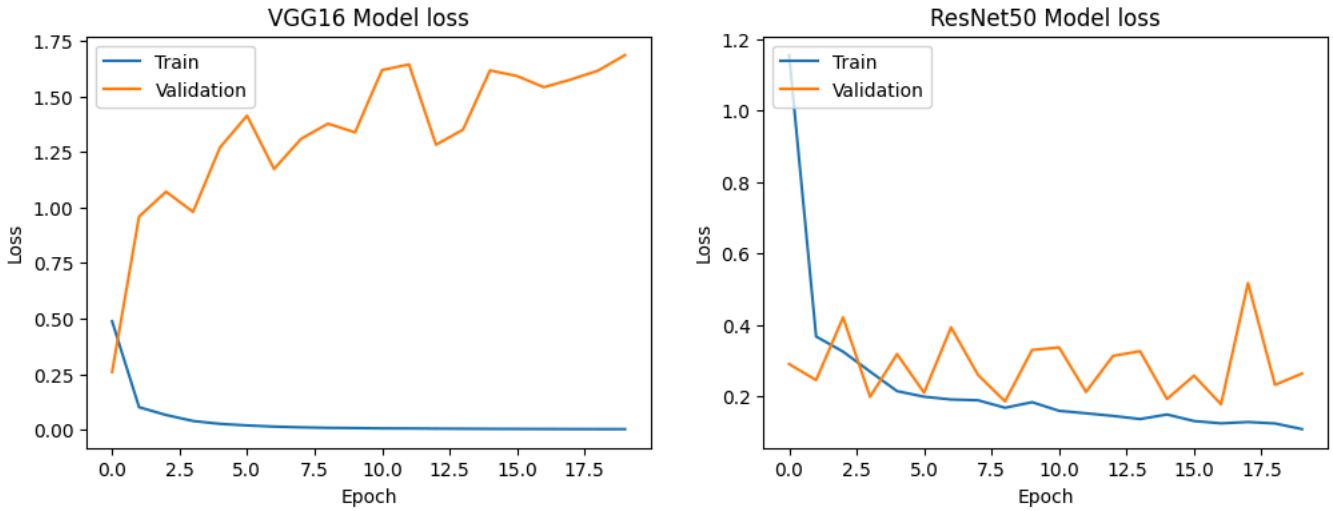
```
#save model - ResNet50
model.save('/content/drive/MyDrive/plant_disease_detection/resnet50_model.h5')
```

Evaluating both models

```
import matplotlib.pyplot as plt


#accuracy
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history_vgg16.history['accuracy'])
plt.plot(history_vgg16.history['val_accuracy'])
plt.title('VGG16 Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.subplot(1, 2, 2)
plt.plot(history_resnet50.history['accuracy'])
plt.plot(history_resnet50.history['val_accuracy'])
plt.title('ResNet50 Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()
```



```
#loss values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history_vgg16.history['loss'])
plt.plot(history_vgg16.history['val_loss'])
plt.title('VGG16 Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.subplot(1, 2, 2)
plt.plot(history_resnet50.history['loss'])
plt.plot(history_resnet50.history['val_loss'])
plt.title('ResNet50 Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()
```




```
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image


#loading models
vgg16_model = load_model('/content/drive/MyDrive/plant_disease_detection/vgg16_model.h5')
resnet50_model = load_model('/content/drive/MyDrive/plant_disease_detection/resnet50_model.h5')
```

```
test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary'
)

# Evaluate models on the test set
loss_vgg16, accuracy_vgg16 = vgg16_model.evaluate(test_generator)
loss_resnet50, accuracy_resnet50 = resnet50_model.evaluate(test_generator)

# Calculate average accuracies over 20 epochs
vgg16_avg_train_accuracy = np.mean(history_vgg16.history['accuracy'])
vgg16_avg_val_accuracy = np.mean(history_vgg16.history['val_accuracy'])
resnet50_avg_train_accuracy = np.mean(history_resnet50.history['accuracy'])
resnet50_avg_val_accuracy = np.mean(history_resnet50.history['val_accuracy'])

# Print the results
print(f"VGG16 Model - Average Training Accuracy (20 Epochs): {vgg16_avg_train_accuracy}")
print(f"VGG16 Model - Average Validation Accuracy (20 Epochs): {vgg16_avg_val_accuracy}")
print(f"VGG16 Model - Test Accuracy: {accuracy_vgg16}")

print(f"ResNet50 Model - Average Training Accuracy (20 Epochs): {resnet50_avg_train_accuracy}")
print(f"ResNet50 Model - Average Validation Accuracy (20 Epochs): {resnet50_avg_val_accuracy}")
print(f"ResNet50 Model - Test Accuracy: {accuracy_resnet50}")
```

```
Found 41 images belonging to 2 classes.
2/2 [==============================] - 22s 5s/step - loss: 0.3225 - accuracy: 0.8293
2/2 [==============================] - 8s 2s/step - loss: 0.2465 - accuracy: 0.9024
VGG16 Model - Average Training Accuracy (20 Epochs): 0.9860169470310212
VGG16 Model - Average Validation Accuracy (20 Epochs): 0.410714291036129
VGG16 Model - Test Accuracy: 0.8292682766914368
ResNet50 Model - Average Training Accuracy (20 Epochs): 0.931779658794403
ResNet50 Model - Average Validation Accuracy (20 Epochs): 0.8892856985330582
ResNet50 Model - Test Accuracy: 0.9024389982223511
```

```
#summarize - VGG16
print("VGG16 Model Summary:")
vgg16_model.summary()
```

```
VGG16 Model Summary:
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 256)               6422784

 dense_1 (Dense)             (None, 1)                 257

=================================================================
Total params: 21137729 (80.63 MB)
Trainable params: 6423041 (24.50 MB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

```
#summarize - ResNet50
print("\nResNet50 Model Summary:")
resnet50_model.summary()
```

```

ResNet50 Model Summary:
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 7, 7, 2048)        23587712

 flatten_1 (Flatten)         (None, 100352)            0

 dense_2 (Dense)             (None, 256)               25690368

 dense_3 (Dense)             (None, 1)                 257
```

```
=================================================================
Total params: 49278337 (187.98 MB)
Trainable params: 25690625 (98.00 MB)
Non-trainable params: 23587712 (89.98 MB)
_________________________________________________________________
```

Testing model with images

```
#preprocessing images
def load_and_preprocess_image(img_path, target_size=(224, 224)):
    img = image.load_img(img_path, target_size=target_size)
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = img_array / 255.0
    return img, img_array


#predicting disease
def predict_disease(model, img_array):
    prediction = model.predict(img_array)
    if prediction[0] < 0.5:
        return 'Diseased'
    else:
        return 'Healthy'


#test image
test_image_path = '/content/drive/MyDrive/plant_disease_detection/test/healthy/1046.JPG.jpeg'
test_image, test_image_array = load_and_preprocess_image(test_image_path)


#VGG16
vgg16_prediction = predict_disease(vgg16_model, test_image_array)

#ResNet50
resnet50_prediction = predict_disease(resnet50_model, test_image_array)
```

```
1/1 [==============================] - 0s 483ms/step
1/1 [==============================] - 0s 184ms/step
```

```
plt.figure(figsize=(2, 2))
plt.imshow(test_image)
plt.title(f'VGG16 Prediction: {vgg16_prediction}\nResNet50 Prediction: {resnet50_prediction}')
plt.axis('off')
plt.show()
```