```
import random
random.seed(0)
import numpy as np
np.random.seed(0)
import tensorflow as tf
tf.random.set_seed(0)
```

```
import os
import json
from zipfile import ZipFile
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models
```

```
!pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.11/dist-packages (1.7.4.2)
Requirement already satisfied: bleach in /usr/local/lib/python3.11/dist-packages (from kaggle) (6.2.0)
Requirement already satisfied: certifi>=14.05.14 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2025.1.31)
Requirement already satisfied: charset-normalizer in /usr/local/lib/python3.11/dist-packages (from kaggle) (3.4.1)
Requirement already satisfied: idna in /usr/local/lib/python3.11/dist-packages (from kaggle) (3.10)
Requirement already satisfied: protobuf in /usr/local/lib/python3.11/dist-packages (from kaggle) (5.29.4)
Requirement already satisfied: python-dateutil>=2.5.3 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.11/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.32.3)
Requirement already satisfied: setuptools>=21.0.0 in /usr/local/lib/python3.11/dist-packages (from kaggle) (75.2.0)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.11/dist-packages (from kaggle) (1.17.0)
Requirement already satisfied: text-unidecode in /usr/local/lib/python3.11/dist-packages (from kaggle) (1.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from kaggle) (4.67.1)
Requirement already satisfied: urllib3>=1.15.1 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.3.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.11/dist-packages (from kaggle) (0.5.1)
```

```
kaggle_credentails = json.load(open("kaggle.json"))
```

```
os.environ['KAGGLE_USERNAME'] = kaggle_credentails["username"]
os.environ['KAGGLE_KEY'] = kaggle_credentails["key"]
```

```
!kaggle datasets download -d abdallahalidev/plantvillage-dataset
```

```
Dataset URL: https://www.kaggle.com/datasets/abdallahalidev/plantvillage-dataset
License(s): CC-BY-NC-SA-4.0
```

```
!ls
```

```
kaggle.json  plantvillage-dataset.zip  sample_data
```

```
with ZipFile("plantvillage-dataset.zip", 'r') as zip_ref:
    zip_ref.extractall()
```

```
print(os.listdir("plantvillage dataset"))
print(len(os.listdir("plantvillage dataset/segmented")))
print(os.listdir("plantvillage dataset/segmented")[:5])
print(len(os.listdir("plantvillage dataset/color")))
print(os.listdir("plantvillage dataset/color")[:5])
print(len(os.listdir("plantvillage dataset/grayscale")))
print(os.listdir("plantvillage dataset/grayscale")[:5])
```

```
['color', 'grayscale', 'segmented']
38
['Grape___Esca_(Black_Measles)', 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot', 'Grape___Black_rot', 'Tomato___Bacterial_spo
38
['Grape___Esca_(Black_Measles)', 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot', 'Grape___Black_rot', 'Tomato___Bacterial_spo
38
['Grape___Esca_(Black_Measles)', 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot', 'Grape___Black_rot', 'Tomato___Bacterial_spo
```

```
print(len(os.listdir("plantvillage dataset/color/Grape___healthy")))
print(os.listdir("plantvillage dataset/color/Grape___healthy")[:5])
```

```
423
['c197dfe9-44d6-4a7e-bb5a-75e2bf05380b___Mt.N.V_HL 6100.JPG', '9ceba66a-d7b0-4ed4-98c3-37d361517a90___Mt.N.V_HL 6147.JPG', 'c05f420
```

```
#DATAPREPROCESSING
base_dir = 'plantvillage dataset/color'
```

```
image_path = '/content/plantvillage dataset/color/Apple___Cedar_apple_rust/025b2b9a-0ec4-4132-96ac-7f2832d0db4a___FREC_C.Rust 3655.JPG'
# Read the image
img = mpimg.imread(image_path)
print(img.shape)
# Display the image
plt.imshow(img)
plt.axis('off')  # Turn off axis numbers
plt.show()
```

```
(256, 256, 3)
```

```
img_size = 224 #resizing the image size because it is having 256,256
batch_size = 32
```

```
data_gen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2  # Use 20% of data for validation
)
```

```
train_generator = data_gen.flow_from_directory(
    base_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    subset='training',
    class_mode='categorical'
)
```

```
Found 43456 images belonging to 38 classes.
```

```
validation_generator = data_gen.flow_from_directory(
    base_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    subset='validation',
    class_mode='categorical'
)
```

```
Found 10849 images belonging to 38 classes.
```

```
#CNN model
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(img_size, img_size, 3)))
model.add(layers.MaxPooling2D(2, 2))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D(2, 2))
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(train_generator.num_classes, activation='softmax'))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_1 (Conv2D)	(None, 109, 109, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 54, 54, 64)	0
flatten (Flatten)	(None, 186624)	0
dense (Dense)	(None, 256)	47,776,000
dense_1 (Dense)	(None, 38)	9,766

Total params: 47,805,158 (182.36 MB)

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

```
# Training the Model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,  # Number of steps per epoch
    epochs=5,  # Number of epochs
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size  # Validation steps
)
```

```
Epoch 1/5
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` c
  self._warn_if_super_not_called()
1358/1358 ━━━━━━━━━━━━━━━━━━━━ 103s 71ms/step - accuracy: 0.6026 - loss: 1.6276 - val_accuracy: 0.8475 - val_loss: 0.4957
Epoch 2/5
1358/1358 ━━━━━━━━━━━━━━━━━━━━ 145s 77ms/step - accuracy: 0.9218 - loss: 0.2513 - val_accuracy: 0.8533 - val_loss: 0.4815
Epoch 3/5
1358/1358 ━━━━━━━━━━━━━━━━━━━━ 133s 71ms/step - accuracy: 0.9683 - loss: 0.1004 - val_accuracy: 0.8909 - val_loss: 0.4031
Epoch 4/5
1358/1358 ━━━━━━━━━━━━━━━━━━━━ 143s 71ms/step - accuracy: 0.9805 - loss: 0.0648 - val_accuracy: 0.8717 - val_loss: 0.5591
```

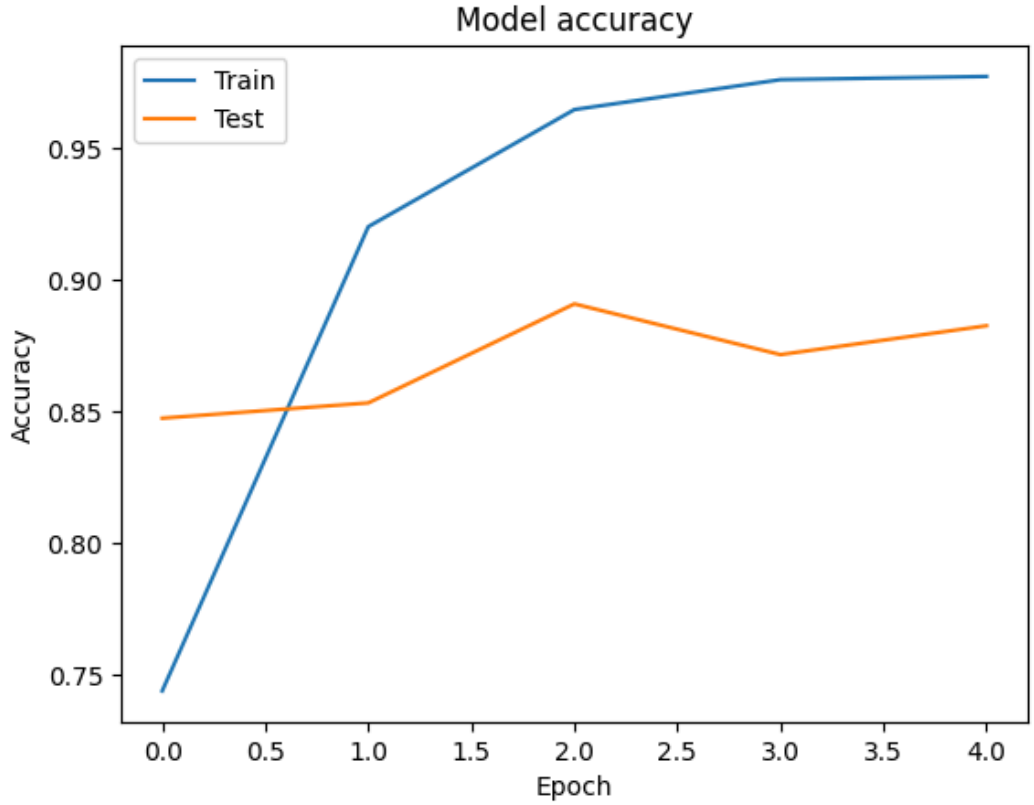
```
Epoch 5/5
1358/1358 ━━━━━━━━━━━━━━━━━━━━ 144s 73ms/step - accuracy: 0.9815 - loss: 0.0569 - val_accuracy: 0.8827 - val_loss: 0.5507
```

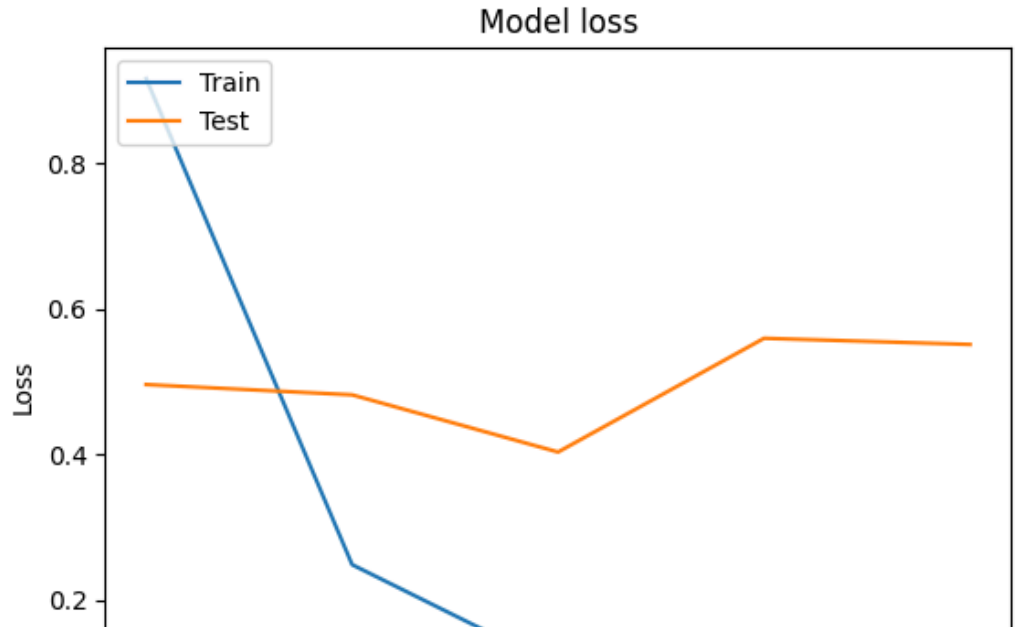
```
print("Evaluating model...")
val_loss, val_accuracy = model.evaluate(validation_generator, steps=validation_generator.samples // batch_size)
print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
```

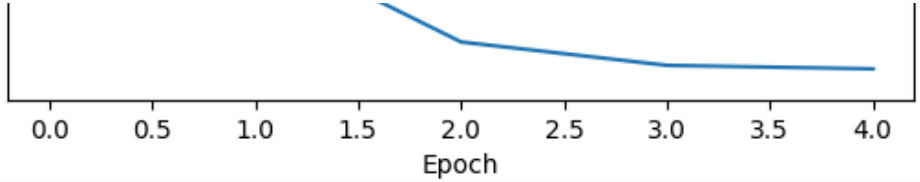
```
Evaluating model...
339/339 ━━━━━━━━━━━━━━━━━━━━ 18s 52ms/step - accuracy: 0.8824 - loss: 0.5475
Validation Accuracy: 88.27%
```

```
# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```


Model accuracy
Train
Test
Accuracy
0.95
0.90
0.85
0.80
0.75
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epoch


Model loss
Train
Test
Loss
0.8
0.6
0.4
0.2




```
from tensorflow.keras.applications import VGG16

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

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_n
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
```

```
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.models import Model

x = base_model.output
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
predictions = Dense(train_generator.num_classes, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_generator, epochs=10, validation_data=validation_generator)
```

```
Epoch 1/10
1358/1358 ━━━━━━━━━━━━━━━━━━━━ 705s 491ms/step - accuracy: 0.3093 - loss: 2.6469 - val_accuracy: 0.6789 - val_loss: 1.0396
Epoch 2/10
1358/1358 ━━━━━━━━━━━━━━━━━━━━ 652s 480ms/step - accuracy: 0.7288 - loss: 0.8699 - val_accuracy: 0.7689 - val_loss: 0.7331
Epoch 3/10
1358/1358 ━━━━━━━━━━━━━━━━━━━━ 680s 500ms/step - accuracy: 0.8202 - loss: 0.5435 - val_accuracy: 0.8273 - val_loss: 0.5322
Epoch 4/10
1358/1358 ━━━━━━━━━━━━━━━━━━━━ 679s 500ms/step - accuracy: 0.8708 - loss: 0.3950 - val_accuracy: 0.8537 - val_loss: 0.4612
Epoch 5/10
1358/1358 ━━━━━━━━━━━━━━━━━━━━ 648s 477ms/step - accuracy: 0.8951 - loss: 0.3150 - val_accuracy: 0.8686 - val_loss: 0.4035
Epoch 6/10
1358/1358 ━━━━━━━━━━━━━━━━━━━━ 645s 475ms/step - accuracy: 0.9169 - loss: 0.2512 - val_accuracy: 0.8914 - val_loss: 0.3523
Epoch 7/10
1358/1358 ━━━━━━━━━━━━━━━━━━━━ 675s 497ms/step - accuracy: 0.9278 - loss: 0.2140 - val_accuracy: 0.8971 - val_loss: 0.3261
```

```
Epoch 8/10
1358/1358 ━━━━━━━━━━━━━━━━━━━━ 671s 494ms/step - accuracy: 0.9311 - loss: 0.2058 - val_accuracy: 0.9052 - val_loss: 0.2959
Epoch 9/10
1358/1358 ━━━━━━━━━━━━━━━━━━━━ 642s 473ms/step - accuracy: 0.9422 - loss: 0.1729 - val_accuracy: 0.8970 - val_loss: 0.3434
Epoch 10/10
1358/1358 ━━━━━━━━━━━━━━━━━━━━ 673s 495ms/step - accuracy: 0.9502 - loss: 0.1496 - val_accuracy: 0.9088 - val_loss: 0.3087
```

```
loss, accuracy = model.evaluate(validation_generator)
print(f"Validation Accuracy: {accuracy * 100:.2f}%")
print(f"Validation Loss: {loss:.4f}")
```

```
340/340 ━━━━━━━━━━━━━━━━━━━━ 51s 150ms/step - accuracy: 0.9111 - loss: 0.3090
Validation Accuracy: 90.88%
Validation Loss: 0.3087
```

```
final_training_accuracy = history.history['accuracy'][-1]  # Last element in the list
print(f"Final Training Accuracy: {final_training_accuracy:.4f}")
```

```
Final Training Accuracy: 0.9460
```

```
def load_and_preprocess_image(image_path, target_size=(224, 224)):
    # Load the image
    img = Image.open(image_path)
    # Resize the image
    img = img.resize(target_size)
    # Convert the image to a numpy array
    img_array = np.array(img)
    # Add batch dimension
    img_array = np.expand_dims(img_array, axis=0)
    # Scale the image values to [0, 1]
    img_array = img_array.astype('float32') / 255.
    return img_array

# Function to Predict the Class of an Image
def predict_image_class(model, image_path, class_indices):
    preprocessed_img = load_and_preprocess_image(image_path)
    predictions = model.predict(preprocessed_img)
    predicted_class_index = np.argmax(predictions, axis=1)[0]
    predicted_class_name = class_indices[predicted_class_index]
    return predicted_class_name
```

```
class_indices = {v: k for k, v in train_generator.class_indices.items()}
```

```
class_indices
```

```
{0: 'Apple___Apple_scab',
 1: 'Apple___Black_rot',
 2: 'Apple___Cedar_apple_rust',
 3: 'Apple___healthy',
 4: 'Blueberry___healthy',
 5: 'Cherry_(including_sour)___Powdery_mildew',
 6: 'Cherry_(including_sour)___healthy',
 7: 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot',
 8: 'Corn_(maize)___Common_rust_',
 9: 'Corn_(maize)___Northern_Leaf_Blight',
 10: 'Corn_(maize)___healthy',
 11: 'Grape___Black_rot',
 12: 'Grape___Esca_(Black_Measles)',
 13: 'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)',
 14: 'Grape___healthy',
 15: 'Orange___Haunglongbing_(Citrus_greening)',
 16: 'Peach___Bacterial_spot',
 17: 'Peach___healthy',
 18: 'Pepper,_bell___Bacterial_spot',
 19: 'Pepper,_bell___healthy',
 20: 'Potato___Early_blight',
 21: 'Potato___Late_blight',
 22: 'Potato___healthy',
 23: 'Raspberry___healthy',
 24: 'Soybean___healthy',
 25: 'Squash___Powdery_mildew',
 26: 'Strawberry___Leaf_scorch',
 27: 'Strawberry___healthy',
 28: 'Tomato___Bacterial_spot',
 29: 'Tomato___Early_blight',
 30: 'Tomato___Late_blight',
 31: 'Tomato___Leaf_Mold',
 32: 'Tomato___Septoria_leaf_spot',
 33: 'Tomato___Spider_mites Two-spotted_spider_mite',
 34: 'Tomato___Target_Spot',
 35: 'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
 36: 'Tomato___Tomato_mosaic_virus',
 37: 'Tomato___healthy'}
```

```
json.dump(class_indices, open('class_indices.json', 'w'))
```

```
image_path = '/content/test_apple_black_rot.JPG'
#image_path = '/content/test_blueberry_healthy.jpg'
#image_path = '/content/test_potato_early_blight.jpg'
predicted_class_name = predict_image_class(model, image_path, class_indices)

# Output the result
print("Predicted Class Name:", predicted_class_name)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 637ms/step
Predicted Class Name: Apple___Black_rot
```