**Seeding for reproducibility**

*# Set seeds for reproducibility*

import random

random.seed(0)

import numpy as np

np.random.seed(0)

import tensorflow as tf

tf.random.set\_seed(0)

**Importing the dependencies**

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

**Data Curation**

Upload the kaggle.json file

!pip install kaggle

Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.5.16)

Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.16.0)

Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from kaggle) (2023.11.17)

Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)

Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.31.0)

Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.66.1)

Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.1)

Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.0.7)

Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle) (6.1.0)

Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle) (0.5.1)

Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle) (1.3)

Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.3.2)

Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.6)

kaggle\_credentails = json.load(open("kaggle.json"))

*# setup Kaggle API key as environment variables*

os.environ['KAGGLE\_USERNAME'] = kaggle\_credentails["username"]

os.environ['KAGGLE\_KEY'] = kaggle\_credentails["key"]

!kaggle datasets download -d abdallahalidev/plantvillage-dataset

Downloading plantvillage-dataset.zip to /content

100% 2.04G/2.04G [00:20<00:00, 202MB/s]

100% 2.04G/2.04G [00:20<00:00, 105MB/s]

!ls

kaggle.json plantvillage-dataset.zip sample\_data

*# Unzip the downloaded dataset*

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])

['grayscale', 'segmented', 'color']

38

['Tomato\_\_\_Spider\_mites Two-spotted\_spider\_mite', 'Tomato\_\_\_Bacterial\_spot', 'Soybean\_\_\_healthy', 'Squash\_\_\_Powdery\_mildew', 'Strawberry\_\_\_healthy']

38

['Tomato\_\_\_Spider\_mites Two-spotted\_spider\_mite', 'Tomato\_\_\_Bacterial\_spot', 'Soybean\_\_\_healthy', 'Squash\_\_\_Powdery\_mildew', 'Strawberry\_\_\_healthy']

38

['Tomato\_\_\_Spider\_mites Two-spotted\_spider\_mite', 'Tomato\_\_\_Bacterial\_spot', 'Soybean\_\_\_healthy', 'Squash\_\_\_Powdery\_mildew', 'Strawberry\_\_\_healthy']

**Number of Classes = 38**

print(len(os.listdir("plantvillage dataset/color/Grape\_\_\_healthy")))

print(os.listdir("plantvillage dataset/color/Grape\_\_\_healthy")[:5])

423

['05cff9d7-0f63-4b6e-9aa3-199cf9ffa64c\_\_\_Mt.N.V\_HL 9111.JPG', '0ce12a10-c6ff-494e-a927-5ddc809c707a\_\_\_Mt.N.V\_HL 8945.JPG', 'c839e134-6692-4065-8e12-7ea01adcc794\_\_\_Mt.N.V\_HL 9014.JPG', '452823c1-22d2-4bce-a4cc-8ad014421261\_\_\_Mt.N.V\_HL 6199.JPG', 'c48a4277-3bd4-45c8-be06-8bdc4404f252\_\_\_Mt.N.V\_HL 6137.JPG']

**Data Preprocessing**

*# Dataset Path*

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)



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)

[[[179 175 176]

[181 177 178]

[184 180 181]

...

[115 112 105]

[108 105 98]

[101 98 91]]

[[176 172 173]

[177 173 174]

[178 174 175]

...

[113 110 103]

[111 108 101]

[109 106 99]]

[[180 176 177]

[180 176 177]

[180 176 177]

...

[108 105 98]

[111 108 101]

[114 111 104]]

...

[[137 128 119]

[131 122 113]

[125 116 107]

...

[ 74 65 48]

[ 74 65 48]

[ 73 64 47]]

[[136 127 118]

[132 123 114]

[128 119 110]

...

[ 77 69 50]

[ 75 67 48]

[ 75 67 48]]

[[133 124 115]

[133 124 115]

[132 123 114]

...

[ 81 73 54]

[ 80 72 53]

[ 79 71 52]]]

*# Image Parameters*

img\_size = 224

batch\_size = 32

**Train Test Split**

*# Image Data Generators*

data\_gen = ImageDataGenerator(

rescale=1./255,

validation\_split=0.2 *# Use 20% of data for validation*

)

*# Train Generator*

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*

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.

**Convolutional Neural Network**

*# Model Definition*

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'))

*# model summary*

model.summary()

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d (Conv2D) (None, 222, 222, 32) 896

max\_pooling2d (MaxPooling2 (None, 111, 111, 32) 0

D)

conv2d\_1 (Conv2D) (None, 109, 109, 64) 18496

max\_pooling2d\_1 (MaxPoolin (None, 54, 54, 64) 0

g2D)

flatten (Flatten) (None, 186624) 0

dense (Dense) (None, 256) 47776000

dense\_1 (Dense) (None, 38) 9766

=================================================================

Total params: 47805158 (182.36 MB)

Trainable params: 47805158 (182.36 MB)

Non-trainable params: 0 (0.00 Byte)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*# Compile the Model*

model.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'])

**Model training**

*# 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

1358/1358 [==============================] - 108s 76ms/step - loss: 0.9791 - accuracy: 0.7328 - val\_loss: 0.4846 - val\_accuracy: 0.8465

Epoch 2/5

1358/1358 [==============================] - 104s 77ms/step - loss: 0.2812 - accuracy: 0.9110 - val\_loss: 0.4477 - val\_accuracy: 0.8655

Epoch 3/5

1358/1358 [==============================] - 106s 78ms/step - loss: 0.1362 - accuracy: 0.9553 - val\_loss: 0.4321 - val\_accuracy: 0.8863

Epoch 4/5

1358/1358 [==============================] - 103s 76ms/step - loss: 0.0891 - accuracy: 0.9708 - val\_loss: 0.5433 - val\_accuracy: 0.8715

Epoch 5/5

1358/1358 [==============================] - 109s 81ms/step - loss: 0.0761 - accuracy: 0.9760 - val\_loss: 0.5091 - val\_accuracy: 0.8828

**Model Evaluation**

*# Model Evaluation*

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 [==============================] - 19s 57ms/step - loss: 0.5091 - accuracy: 0.8828

Validation Accuracy: 88.28%

*# 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'])

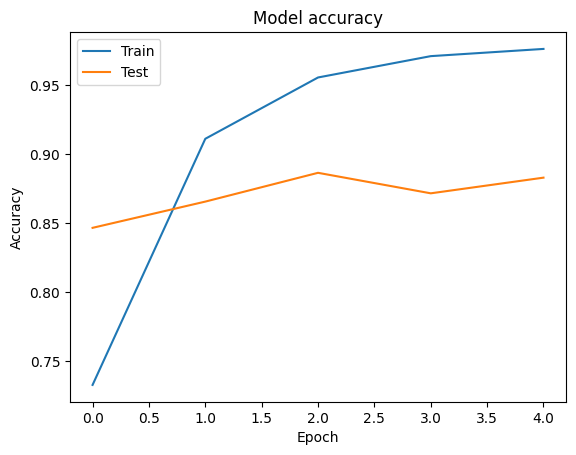
plt.title('Model loss')

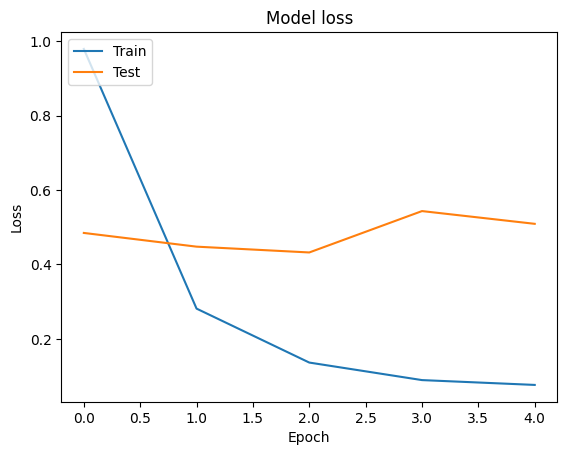
plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()





**Building a Predictive System**

*# Function to Load and Preprocess the Image using Pillow*

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

*# Create a mapping from class indices to class names*

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'}

*# saving the class names as json file*

json.dump(class\_indices, open('class\_indices.json', 'w'))

*# Example Usage*

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 [==============================] - 0s 266ms/step

Predicted Class Name: Apple\_\_\_Black\_rot

**Save the model to Google drive or local**

model.save('drive/MyDrive/Youtube/trained\_models/plant\_disease\_prediction\_model.h5')

model.save('plant\_disease\_prediction\_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 `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my\_model.keras')`.

saving\_api.save\_model(