**Code Explanation**

## **Import Libraries**

**import matplotlib.pyplot as plt**

**import numpy as np**

**import PIL**

**import tensorflow as tf**

**import os**

**from tensorflow import keras**

**from tensorflow.keras import layers**

**from tensorflow.keras.preprocessing.image import ImageDataGenerator**

**from tensorflow.keras.models import Sequential**

matplotlib.pyplot is a sub-library of the matplotlib library that provides functions for creating visualizations of data. plt is a commonly used alias for this sublibrary.

numpy is a library for working with arrays and numerical data. It is used here with the alias np.

PIL is the Python Imaging Library, which provides functions for working with images.

tensorflow is a machine learning library developed by Google. It provides a wide range of tools for building and training machine learning models. tf is a commonly used alias for this library.

keras is a high-level API for building and training machine learning models. It runs on top of tensorflow, and can be used to define, compile, and train models.

layers is a module in the keras library that provides functions for building neural network layers.

ImageDataGenerator is a class in the keras.preprocessing.image module that can be used to generate batches of image data for training machine learning models.

Sequential is a class in the keras.models module that represents a linear stack of layers in a neural network.

## **Load Dataset and Preprocess**

batch\_size = 32

img\_height = 150

img\_width = 150

train\_datagen = ImageDataGenerator(

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True,

validation\_split=0.2)

train\_generator = train\_datagen.flow\_from\_directory(

'PlantVillage',

target\_size=(150, 150),

batch\_size=32,

class\_mode='categorical',

subset='training')

validation\_generator = train\_datagen.flow\_from\_directory(

'PlantVillage',

target\_size=(150, 150),

batch\_size=32,

class\_mode='categorical',

subset='validation')

This code sets up an ImageDataGenerator object to generate batches of image data for training a machine learning model.

The batch\_size variable specifies the number of images to include in each batch.

The img\_height and img\_width variables specify the dimensions to which the images will be resized.

The train\_datagen object is created using the ImageDataGenerator class, with several arguments passed to the constructor:

* shear\_range: a float that specifies the range for randomly applying shearing transformations.
* zoom\_range: a float that specifies the range for randomly zooming in on the images.
* horizontal\_flip: a boolean that specifies whether to randomly flip the images horizontally.
* validation\_split: a float between 0 and 1 that specifies the proportion of the data to use for validation.

The train\_generator and validation\_generator variables are created using the flow\_from\_directory method of the train\_datagen object. This method generates batches of image data from the specified directory.

The target\_size argument specifies the dimensions to which the images should be resized.

The class\_mode argument specifies the format in which the data should be returned. The value 'categorical' indicates that the data should be returned in a one-hot encoded format.

The subset argument specifies which portion of the data to use. The value 'training' specifies that the train\_generator should use the training portion of the data, while the value 'validation' specifies that the validation\_generator should use the validation portion of the data.

## **Design Model**

num\_classes=15

model = Sequential([

layers.RescalingRescaling(1./255, input\_shape=(img\_height, img\_width, 3)),

layers.Conv2D(16, 3, padding='same', activation='relu'),

layers.MaxPooling2D(),

layers.Conv2D(32, 3, padding='same', activation='relu'),

layers.MaxPooling2D(),

layers.Conv2D(64, 3, padding='same', activation='relu'),

layers.MaxPooling2D(),

layers.Flatten(),

layers.Dense(128, activation='relu'),

layers.Dense(num\_classes)

])

model.compile(optimizer='adam',

loss=tf.keras.losses.CategoricalCrossentropy(from\_logits=True),

metrics=['accuracy'])

This code defines and compiles a convolutional neural network (CNN) using the keras library.

The num\_classes variable specifies the number of classes in the dataset.

The model variable is created using the Sequential class, which represents a linear stack of layers in a neural network. The layers are specified in a list passed as an argument to the Sequential constructor.

The layers of the model are as follows:

* A Rescaling layer, which scales the input data by dividing it by 255. This is typically done to normalize the pixel values of the input images to the range [0, 1].
* A Conv2D layer with 16 filters, a kernel size of 3, and the 'same' padding. The 'relu' activation function is used. This layer applies 2D convolution to the input data and reduces the spatial dimensions (i.e., width and height) through the use of max pooling.
* A MaxPooling2D layer, which reduces the spatial dimensions of the data through the use of max pooling.
* A Conv2D layer with 32 filters, a kernel size of 3, and the 'same' padding. The 'relu' activation function is used. This layer applies 2D convolution to the input data and reduces the spatial dimensions through max pooling.
* A MaxPooling2D layer, which reduces the spatial dimensions of the data through max pooling.
* A Conv2D layer with 64 filters, a kernel size of 3, and the 'same' padding. The 'relu' activation function is used. This layer applies 2D convolution to the input data and reduces the spatial dimensions through max pooling.
* A MaxPooling2D layer, which reduces the spatial dimensions of the data through max pooling.
* A Flatten layer, which flattens the data into a one-dimensional array.
* A Dense layer with 128 units and the 'relu' activation function. This layer is a fully-connected layer that processes the flattened data.
* A Dense layer with num\_classes units. This layer is a fully-connected layer that produces the output of the model.

The model is compiled using the compile method, with the following arguments:

* optimizer: the optimization algorithm to use. The value 'adam' specifies the Adam optimization algorithm.
* loss: the loss function to use. The CategoricalCrossentropy loss is used here, with the from\_logits argument set to True. This loss function is used when the model's output are logits (i.e., unnormalized predictions) and the classes are mutually exclusive (e.g., each image belongs to exactly one class).
* metrics: a list of metrics to use for evaluating the model. The value ['accuracy'] specifies that the model's accuracy should be used as a metric.

## **Train Model**

history=model.fit(

train\_generator,

epochs=15,

validation\_data=validation\_generator)

This code trains the CNN defined in the previous code block using the image data generated by the train\_generator and validation\_generator objects.

The fit method of the model object is used to train the model on the data. The following arguments are passed to the fit method:

* train\_generator: the generator that generates the training data.
* epochs: the number of epochs (i.e., iterations) to train the model for.
* validation\_data: the generator that generates the validation data.

The history variable is assigned the return value of the fit method, which is a History object that contains information about the training process, including the loss and metrics values at the end of each epoch. This information can be used to visualize the training process and evaluate the model's performance.

## **Test Model**

import random

import cv2 as cv

import tensorflow as tf

# Recreate the exact same model, including its weights and the optimizer

model\_test = tf.keras.models.load\_model('model2.hdf5')

dir = "PlantVillage"

sub\_dir = random.sample(os.listdir(dir), k=1)[0]

image = random.sample(os.listdir(dir+"/"+sub\_dir), k=1)[0]

print("Load Image: ", dir+"/"+sub\_dir+"/"+image)

data = cv.imread(dir+"/"+sub\_dir+"/"+image)

data = cv.cvtColor(data, cv.COLOR\_BGR2RGB)

# Show Image

plt.figure()

plt.imshow(data)

plt.title(sub\_dir)

plt.show()

# Preprocess Data

im\_res = cv.resize(data, (150, 150))

im\_res = np.reshape(im\_res, [1, 150,150,3])

# Prediction

result = model\_test.predict(im\_res)

predicted\_class = sorted(os.listdir(dir))[np.argmax(result[0])]

print("Predicted Class: ", predicted\_class)

print("Actual Class: ", sub\_dir)

This code loads a trained CNN model from a file and uses it to make a prediction on a randomly selected image from a directory.

The model is loaded using the load\_model function from the keras.models module, which takes the file path of the saved model as an argument.

The dir variable specifies the directory containing the images. A subdirectory is randomly selected using the random.sample function and the os.listdir function, which lists the contents of a directory. Then, a randomly selected image from the subdirectory is loaded using the random.sample function and the os.listdir function.

The image is displayed using the imshow function from the pyplot module of the matplotlib library.

The image is preprocessed by resizing it to the dimensions specified in the model (150x150) using the resize function from the cv2 library, and reshaping it into the shape expected by the model using the reshape function from numpy.

The model's predict method is used to make a prediction on the preprocessed image. The predicted class is determined by finding the index of the highest element in the output array, and looking up the corresponding class name in the list of class names obtained using os.listdir. The actual class of the image is also displayed for comparison.