```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
```

```
!kaggle datasets download -d emmarex/plantdisease
```

```
    Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
    Downloading plantdisease.zip to /content
    100% 655M/658M [00:31<00:00, 22.9MB/s]
    100% 658M/658M [00:31<00:00, 22.1MB/s]
```

```
import zipfile
zip_ref = zipfile.ZipFile('/content/plantdisease.zip', 'r')
zip_ref.extractall('/content')
zip_ref.close()
```

```
!pip install --upgrade keras
```

```
    Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
    Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (2.12.0)
```

```
import numpy as np
import pickle
import cv2
from os import listdir
from sklearn.preprocessing import LabelBinarizer
from keras.models import Sequential
import tensorflow as tf
tf.keras.layers.BatchNormalization()
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation, Flatten, Dropout, Dense
from keras import backend as K
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
from keras.preprocessing import image
from tensorflow.keras.utils import img_to_array
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

```
EPOCHS = 50
INIT_LR = 1e-3
BS = 32
default_image_size = tuple((256, 256))
image_size = 0
directory_root = '/content/plantvillage'
width=256
height=256
depth=3
```

```
def convert_image_to_array(image_dir):
    try:
        image = cv2.imread(image_dir)
        if image is not None :
            image = cv2.resize(image, default_image_size)
            return img_to_array(image)
        else :
            return np.array([])
    except Exception as e:
        print(f"Error : {e}")
        return None
```

```
image_list, label_list = [], []
try:
    print("[INFO] Loading images ...")
    root_dir = listdir(directory_root)
    for directory in root_dir:
        # remove .DS_Store from list
        if directory == ".DS_Store":
            root_dir.remove(directory)

    for plant_folder in root_dir:
        plant_disease_folder_list = listdir(f"{directory_root}/{plant_folder}")

        for disease_folder in plant_disease_folder_list:
            # remove .DS_Store from list
            if disease_folder == ".DS_Store":
                plant_disease_folder_list.remove(disease_folder)

        for plant_disease_folder in plant_disease_folder_list:
            print(f"[INFO] Processing {plant_disease_folder} ...")
            plant_disease_image_list = listdir(f"{directory_root}/{plant_folder}/{plant_disease_folder}/")

            for single_plant_disease_image in plant_disease_image_list:
                if single_plant_disease_image == ".DS_Store":
                    plant_disease_image_list.remove(single_plant_disease_image)

            for image_file in plant_disease_image_list[:200]:
                image_directory = f"{directory_root}/{plant_folder}/{plant_disease_folder}/{image_file}"
                if image_file.endswith(".jpg") or image_file.endswith(".JPG"):
                    image_list.append(convert_image_to_array(image_directory))
                    label_list.append(plant_disease_folder)
    print("[INFO] Image loading completed")
except Exception as e:
    print(f"Error: {e}")
```

```
[INFO] Loading images ...
[INFO] Processing Tomato_Bacterial_spot ...
[INFO] Processing Tomato_Septoria_leaf_spot ...
[INFO] Processing Tomato_Spider_mites_Two_spotted_spider_mite ...
[INFO] Processing Potato___Late_blight ...
[INFO] Processing Tomato__Tomato_mosaic_virus ...
[INFO] Processing Potato___Early_blight ...
[INFO] Processing Pepper__bell___Bacterial_spot ...
[INFO] Processing Tomato__Target_Spot ...
[INFO] Processing Tomato__Tomato_YellowLeaf__Curl_Virus ...
[INFO] Processing Pepper__bell___healthy ...
[INFO] Processing Potato___healthy ...
[INFO] Processing Tomato_Early_blight ...
[INFO] Processing Tomato_Leaf_Mold ...
[INFO] Processing Tomato_Late_blight ...
[INFO] Processing Tomato_healthy ...
[INFO] Image loading completed
```

```
image_size = len(image_list)
```

```
label_binarizer = LabelBinarizer()
image_labels = label_binarizer.fit_transform(label_list)
```

```
pickle.dump(label_binarizer,open('label_transform.pkl', 'wb'))
n_classes = len(label_binarizer.classes_)
```

```
print(label_binarizer.classes_)
```

```
['Pepper__bell___Bacterial_spot' 'Pepper__bell___healthy'
 'Potato___Early_blight' 'Potato___Late_blight' 'Potato___healthy'
 'Tomato_Bacterial_spot' 'Tomato_Early_blight' 'Tomato_Late_blight'
 'Tomato_Leaf_Mold' 'Tomato_Septoria_leaf_spot'
 'Tomato_Spider_mites_Two_spotted_spider_mite' 'Tomato__Target_Spot'
 'Tomato__Tomato_YellowLeaf__Curl_Virus' 'Tomato__Tomato_mosaic_virus'
 'Tomato_healthy']
```

```
np_image_list = np.array(image_list, dtype=np.float16) / 225.0
```

```
print("[INFO] Spliting data to train, test")
x_train, x_test, y_train, y_test = train_test_split(np_image_list, image_labels, test_size=0.2, random_state = 42)
```

```
[INFO] Spliting data to train, test
```

```
aug = ImageDataGenerator(
    rotation_range=25, width_shift_range=0.1,
    height_shift_range=0.1, shear_range=0.2,
    zoom_range=0.2,horizontal_flip=True,
    fill_mode="nearest")
```

```
from keras.layers import BatchNormalization
model = Sequential()
inputShape = (height, width, depth)
chanDim = -1
if K.image_data_format() == "channels_first":
    inputShape = (depth, height, width)
    chanDim = 1
model.add(Conv2D(32, (3, 3), padding="same",input_shape=inputShape))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(1024))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(n_classes))
model.add(Activation("softmax"))
```

```
model.summary()
```

```
 conv2d_1 (Conv2D)           (None, 85, 85, 64)        18496

 activation_1 (Activation)   (None, 85, 85, 64)        0

 batch_normalization_2 (Batc  (None, 85, 85, 64)       256
 hNormalization)

 conv2d_2 (Conv2D)           (None, 85, 85, 64)        36928

 activation_2 (Activation)   (None, 85, 85, 64)        0

 batch_normalization_3 (Batc  (None, 85, 85, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 42, 42, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 42, 42, 64)        0

 conv2d_3 (Conv2D)           (None, 42, 42, 128)       73856

 activation_3 (Activation)   (None, 42, 42, 128)       0
 flatten (Flatten)           (None, 56448)             0

 dense (Dense)               (None, 1024)              57803776

 activation_5 (Activation)   (None, 1024)              0

 batch_normalization_6 (Batc  (None, 1024)             4096
 hNormalization)

 dropout_3 (Dropout)         (None, 1024)              0

 dense_1 (Dense)             (None, 15)                15375

 activation_6 (Activation)   (None, 15)                0

=================================================================
Total params: 58,102,671
Trainable params: 58,099,791
Non-trainable params: 2,880
_________________________________________________________________
```

```
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
# distribution
model.compile(loss="binary_crossentropy", optimizer=opt,metrics=["accuracy"])
# train the network
print("[INFO] training network...")
```

```
[INFO] training network...
/usr/local/lib/python3.10/dist-packages/keras/optimizers/legacy/adam.py:117: UserWarning: The `lr` argument is deprecated, use `learnin
  super().__init__(name, **kwargs)
```

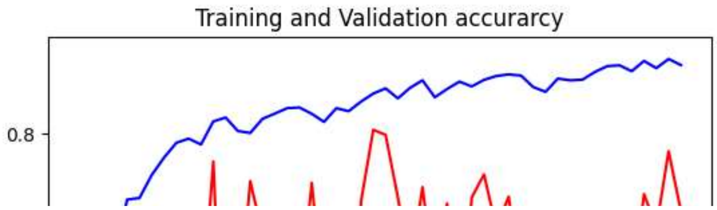
```
history = model.fit_generator(
    aug.flow(x_train, y_train, batch_size=BS),
    validation_data=(x_test, y_test),
    steps_per_epoch=len(x_train) // BS,
    epochs=EPOCHS, verbose=1
    )
    Epoch 44/50
    73/73 [==============================] - 35s 479ms/step - loss: 0.0339 - accuracy: 0.9326 - val_loss: 0.5219 - val_accuracy: 0.4856
    Epoch 45/50
    73/73 [==============================] - 35s 485ms/step - loss: 0.0330 - accuracy: 0.9343 - val_loss: 0.2857 - val_accuracy: 0.5736
    Epoch 46/50
    73/73 [==============================] - 34s 467ms/step - loss: 0.0360 - accuracy: 0.9227 - val_loss: 0.4447 - val_accuracy: 0.4839
    Epoch 47/50
    73/73 [==============================] - 36s 491ms/step - loss: 0.0292 - accuracy: 0.9429 - val_loss: 0.2345 - val_accuracy: 0.6802
    Epoch 48/50
    73/73 [==============================] - 36s 488ms/step - loss: 0.0355 - accuracy: 0.9287 - val_loss: 0.2063 - val_accuracy: 0.6142
    Epoch 49/50
    73/73 [==============================] - 36s 486ms/step - loss: 0.0284 - accuracy: 0.9468 - val_loss: 0.1321 - val_accuracy: 0.7648
    Epoch 50/50
    73/73 [==============================] - 36s 488ms/step - loss: 0.0324 - accuracy: 0.9347 - val_loss: 0.2505 - val_accuracy: 0.6464
```

```
print(history.history.keys())
```

```
    dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(accuracy) + 1)
#Train and validation accuracy
plt.plot(epochs, accuracy, 'b', label='Training accurarcy')
plt.plot(epochs, val_accuracy, 'r', label='Validation accurarcy')
plt.title('Training and Validation accurarcy')
plt.legend()

plt.figure()
#Train and validation loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
```



```
print("[INFO] Calculating model accuracy")
scores = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {scores[1]*100}")
```

```
    [INFO] Calculating model accuracy
    19/19 [==============================] - 1s 34ms/step - loss: 0.2505 - accuracy: 0.6464
    Test Accuracy: 64.63621258735657
```

```
print("[INFO] Saving model...")
pickle.dump(model,open('cnn_model1.pkl', 'wb'))
```

```
    [INFO] Saving model...
```

```
import pickle

# Assuming `model` is your Keras model
with open('/content/cnn_model1.pkl', 'wb') as file:
    pickle.dump(model, file)
```

```
# Assuming `model` is your Keras model
model.save('/content/cnn_model1.h5')
```

```
from tensorflow.keras.models import load_model

loaded_model = load_model('/content/cnn_model1.h5')
```

```
from PIL import Image
import numpy as np
from tensorflow.keras.preprocessing.image import img_to_array

image_dir = "/content/Potato___Early_blight.jpeg"

# Load the image using PIL
image = Image.open(image_dir)

# Convert the image to a NumPy array
np_image = img_to_array(image)

# Normalize the image
np_image = np_image.astype('float32') / 255.0

# Expand dimensions to create a batch of size 1
np_image = np.expand_dims(np_image, axis=0)
```

```
import cv2
import tensorflow as tf

# Load the trained model
loaded_model = tf.keras.models.load_model('/content/cnn_model1.h5')

# Define the desired size for resizing
target_size = (256, 256)

# Create an empty destination image with the desired size
resized_image = np.empty((1, *target_size, 3), dtype=np.float32)

# Resize the image to match the desired size
resized_image[0] = cv2.resize(np_image[0], target_size)

# Make predictions
```

```
result = loaded_model.predict(resized_image)

print(result)
```

```
1/1 [==============================] - 1s 633ms/step
[[1.9801724e-36 2.4690417e-28 7.6511581e-21 6.1925682e-31 0.0000000e+00
  4.1863404e-34 9.5526785e-01 2.2288880e-09 2.3369914e-20 4.4732232e-02
  3.3458307e-16 4.0562029e-22 0.0000000e+00 2.2592170e-24 0.0000000e+00]]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-30-0d3832c0ead6> in <cell line: 4>()
      2 
      3 
----> 4 result=model.predict(np_image)
      5 print(result)

                              1 frames
/usr/local/lib/python3.10/dist-packages/keras/engine/training.py in
tf__predict_function(iterator)
     13                 try:
     14                     do_return = True
---> 15                     retval_ = ag__.converted_call(ag__.ld(step_function),
(ag__.ld(self), ag__.ld(iterator)), None, fscope)
     16                 except:
     17                     do_return = False

ValueError: in user code:

    File "/usr/local/lib/python3.10/dist-packages/keras/engine/training.py", line
2169, in predict_function  *
        return step_function(self, iterator)
    File "/usr/local/lib/python3.10/dist-packages/keras/engine/training.py", line
2155, in step_function  **
        outputs = model.distribute_strategy.run(run_step, args=(data,))
    File "/usr/local/lib/python3.10/dist-packages/keras/engine/training.py", line
2143, in run_step  **
        outputs = model.predict_step(data)
    File "/usr/local/lib/python3.10/dist-packages/keras/engine/training.py", line
2111, in predict_step
        return self(x, training=False)
    File "/usr/local/lib/python3.10/dist-packages/keras/utils/traceback_utils.py",
line 70, in error_handler
        raise e.with_traceback(filtered_tb) from None
    File "/usr/local/lib/python3.10/dist-packages/keras/engine/input_spec.py", line
```

```
# try this after running all
import cv2
import tensorflow as tf

# Load the trained model
loaded_model = tf.keras.models.load_model('/content/cnn_model1.h5')

# Define the desired size for resizing
target_size = (256, 256)

# Create an empty destination image with the desired size
resized_image = np.empty((1, *target_size, 3), dtype=np.float32)

# Resize the image to match the desired size
resized_image[0] = cv2.resize(np_image[0], target_size)

# Make predictions
result = loaded_model.predict(resized_image)

# Retrieve the class label
class_index = np.argmax(result)
class_label = label_binarizer.classes_[class_index]

print("Predicted class:", class_label)
```

```
1/1 [==============================] - 0s 175ms/step
Predicted class: Tomato_Early_blight
```

✓ 5s completed at 12:26 PM ● ×