```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt


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

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

```
IMAGE_SIZE = 256
BATCH_SIZE = 32
CHANNELS = 3
EPOCHS = 50

dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "/content/drive/MyDrive/PlantVillage",
    shuffle = True,
    image_size = (IMAGE_SIZE,IMAGE_SIZE),
    batch_size = BATCH_SIZE
)
```

```
Found 2152 files belonging to 3 classes.
```

```
class_names = dataset.class_names
class_names
```

```
['Potato___Early_blight', 'Potato___Late_blight', 'Potato___healthy']
```

```
len(dataset)
```

```
68
```

```
for image_batch, label_batch in dataset.take(1):
    print(image_batch.shape)
    print(label_batch.numpy())
```

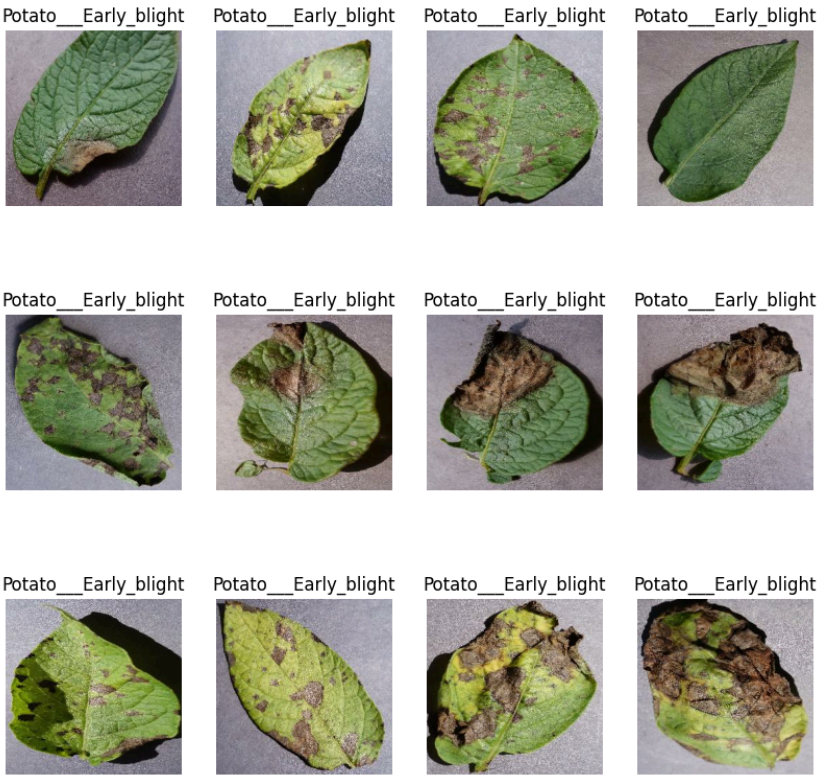
```
(32, 256, 256, 3)
 [0 1 1 1 0 1 0 1 0 1 1 0 1 0 1 1 0 0 1 2 1 0 0 0 0 1 0 1 0 2 1 1]
```

```
for image_batch, label_batch in dataset.take(1):
    # without .numpy() functio it give tensor

    print(image_batch[0].numpy())
    # image_batch[0].numpy() it gives 3D array

    # these values are float we have to convert them into int using astype() functi
    plt.imshow(image_batch[0].numpy().astype("uint8"))
    plt.axis("off")
```

```
[[[189. 180. 183.]
  [190. 181. 184.]
  [193. 184. 187.]
  ...
  [162. 155. 162.]
  [159. 152. 159.]
  [157. 150. 157.]]

 [[186. 177. 180.]
  [186. 177. 180.]
  [187. 178. 181.]
  ...
  [161. 154. 161.]
  [159. 152. 159.]
  [158. 151. 158.]]

 [[184. 175. 178.]
  [182. 173. 176.]
  [182. 173. 176.]
  ...
  [159. 152. 159.]
  [159. 152. 159.]
  [159. 152. 159.]]

 ...

 [[151. 140. 138.]
  [145. 134. 132.]
  [159. 148. 146.]
  ...
  [110.  99. 105.]
  [127. 116. 122.]
  [143. 132. 138.]]

 [[153. 142. 140.]
  [150. 139. 137.]
  [156. 145. 143.]
  ...
  [109.  98. 104.]
  [127. 116. 122.]
  [138. 127. 133.]]

 [[155. 144. 142.]
  [153. 142. 140.]
  [144. 133. 131.]
  ...
  [120. 109. 115.]
  [125. 114. 120.]
  [119. 108. 114.]]]
```

```
# plt.figure(figsize = (10,10)) WE USED IT SO THE IMAGES DONT OVERLAP BELOW
plt.figure(figsize = (10,10))
for image_batch, label_batch in dataset.take(1):
```

```
    # we onnly took 12 images from teh dataset to print beacuse 32 will ake it diff
    for i in range(12):
        # subplot help us to plot multiple images at a time
        ax = plt.subplot(3,4,i+1)

        # without .numpy() functio it give tensor
        # these values are float we have to convert them into int using astype() fu
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[label_batch[1]])
        plt.axis("off")
```




```
len(dataset)
```

```
68
```

80% ==> training

20% ==> 10% validation, 10% testing

```
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1,
    assert (train_split + test_split + val_split) == 1

    ds_size = len(ds)

    if shuffle:
        # seed is just for predictibilty, if you do same seed everytime you get s
        ds = ds.shuffle(shuffle_size, seed=12)

    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)
```

```
    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds


train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)


len(train_ds)
```

```
54
```

```
len(val_ds)
```

```
6
```

```
len(test_ds)
```

```
8
```

```
# cache() it will read the image from the disk and then for the next iteration it
# prefetch(), if you are using GPU and CPU, if GPU is busy training prefetch will
# buffer_size = tf.data.AUTOTUNE using this i am letting tensorflow determine how

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size = tf.data.AUTOTUNE
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size = tf.data.AUTOTUNE)
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size = tf.data.AUTOTUNE)


# Now after the above step my dataset are kind of optimized for training performa
```

```
resize_and_rescale = tf.keras.Sequential([
    # here we did resizing using this api
    layers.Resizing(IMAGE_SIZE, IMAGE_SIZE),

    # here we have done scaling of the image with this api
    layers.Rescaling(1.0/255)
])


data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.2)
])


input_shape = (IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
# print(input_shape)
n_classes = 3


model = models.Sequential([
    # resize_and_rescale,
    layers.Input(shape=input_shape),

    # Convolutional Layers
    layers.Conv2D(32, kernel_size = (3,3), activation='relu'),
    # layers.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape=input
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
```

```
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    # Neural Network
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax'),

])

model.build(input_shape = (32,input_shape))


model.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 254, 254, 32)	896
max_pooling2d_7 (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_8 (Conv2D)	(None, 125, 125, 64)	18,496
max_pooling2d_8 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_9 (Conv2D)	(None, 60, 60, 64)	36,928
max_pooling2d_9 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_10 (Conv2D)	(None, 28, 28, 64)	36,928
max_pooling2d_10 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_11 (Conv2D)	(None, 12, 12, 64)	36,928
max_pooling2d_11 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_12 (Conv2D)	(None, 4, 4, 64)	36,928
max_pooling2d_12 (MaxPooling2D)	(None, 2, 2, 64)	0
flatten_2 (Flatten)	(None, 256)	0
dense_4 (Dense)	(None, 64)	16,448
dense_5 (Dense)	(None, 3)	195

Total params: 183,747 (717.76 KB)
Trainable params: 183,747 (717.76 KB)

Start coding or generate with AI.

```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)


history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=25,
)
```

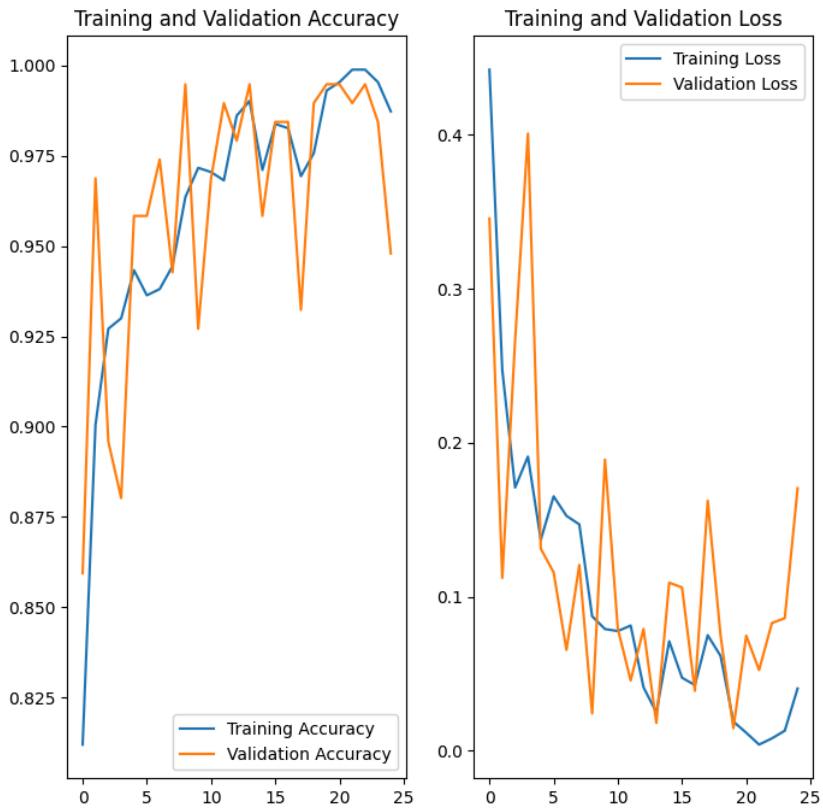
```
Epoch 1/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 51ms/step - accuracy: 0.7755 - loss: 0.5215 - v
Epoch 2/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 50ms/step - accuracy: 0.8804 - loss: 0.2846 - v
Epoch 3/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 50ms/step - accuracy: 0.9243 - loss: 0.1778 - v
```

```
Epoch 4/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 50ms/step - accuracy: 0.9431 - loss: 0.1658 - v
Epoch 5/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 49ms/step - accuracy: 0.9282 - loss: 0.1757 - v
Epoch 6/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 58ms/step - accuracy: 0.9525 - loss: 0.1192 - v
Epoch 7/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 5s 50ms/step - accuracy: 0.9326 - loss: 0.1541 - v
Epoch 8/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 50ms/step - accuracy: 0.9563 - loss: 0.1162 - v
Epoch 9/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 50ms/step - accuracy: 0.9512 - loss: 0.1058 - v
Epoch 10/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 51ms/step - accuracy: 0.9811 - loss: 0.0526 - v
Epoch 11/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 5s 50ms/step - accuracy: 0.9613 - loss: 0.1101 - v
Epoch 12/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 5s 51ms/step - accuracy: 0.9801 - loss: 0.0537 - v
Epoch 13/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 50ms/step - accuracy: 0.9884 - loss: 0.0378 - v
Epoch 14/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 5s 51ms/step - accuracy: 0.9914 - loss: 0.0187 - v
Epoch 15/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 50ms/step - accuracy: 0.9768 - loss: 0.0582 - v
Epoch 16/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 51ms/step - accuracy: 0.9849 - loss: 0.0409 - v
Epoch 17/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 5s 52ms/step - accuracy: 0.9777 - loss: 0.0560 - v
Epoch 18/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 51ms/step - accuracy: 0.9757 - loss: 0.0598 - v
Epoch 19/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 50ms/step - accuracy: 0.9690 - loss: 0.0870 - v
Epoch 20/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - accuracy: 0.9965 - loss: 0.0143 - v
Epoch 21/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 56ms/step - accuracy: 0.9922 - loss: 0.0174 - v
Epoch 22/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 51ms/step - accuracy: 0.9998 - loss: 0.0038 - v
Epoch 23/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 50ms/step - accuracy: 0.9980 - loss: 0.0130 - v
Epoch 24/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 49ms/step - accuracy: 0.9974 - loss: 0.0080 - v
Epoch 25/25
54/54 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - accuracy: 0.9977 - loss: 0.0084 - v
```

```
scores = model.evaluate(test_ds)
scores
```

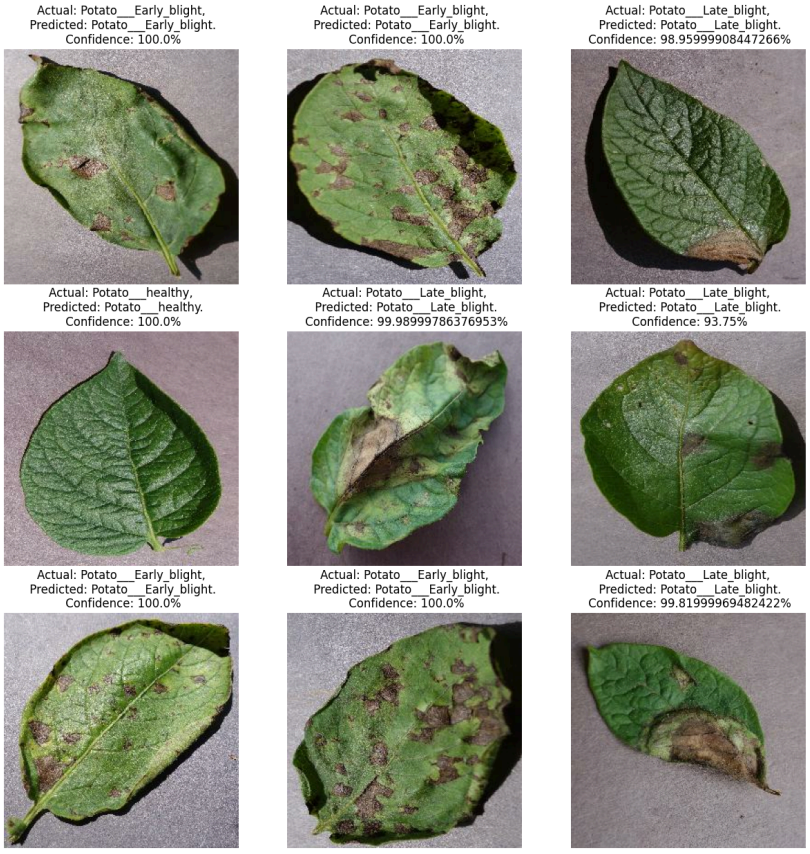
```
8/8 ━━━━━━━━━━━━━━━━━━━━ 6s 21ms/step - accuracy: 0.9375 - loss: 0.2064
[0.19714942574501038, 0.9453125]
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']


EPOCHS = 25
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(EPOCHS), acc, label='Training Accuracy')
plt.plot(range(EPOCHS), val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(EPOCHS), loss, label='Training Loss')
plt.plot(range(EPOCHS), val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```




```
def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array, 0)

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence


import numpy as np
plt.figure(figsize=(15, 15))
for images, labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class, confidence = predict(model, images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n Con

        plt.axis("off")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
```




```
model.save(".\model.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
```