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

```
     Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from keras.layers import *
from keras.models import *
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
import os, shutil
import warnings
warnings.filterwarnings('ignore')
```

```
# Let's plot a few images
train_path = "/content/drive/MyDrive/Vegetable Images/train"
validation_path = "../content/drive/MyDrive/Vegetable Images/validation"
test_path = "../content/drive/MyDrive/Vegetable Images/test"

image_categories = os.listdir('../content/drive/MyDrive/Vegetable Images/train')

def plot_images(image_categories):

    # Create a figure
    plt.figure(figsize=(12, 12))
    for i, cat in enumerate(image_categories):
       image_path = train_path + '/' + cat
       images_in_folder = os.listdir(image_path)
       first_image_of_folder = images_in_folder[0]
       first_image_path = image_path + '/' + first_image_of_folder
       img = image.load_img(first_image_path)
       img_arr = image.img_to_array(img)/255.0
       plt.subplot(4, 4, i+1)
       plt.imshow(img_arr)
       plt.title(cat)
       plt.axis('off')
       plt.show()

# Call the function
plot_images(image_categories)
```

Pumpkin

Radish

Papaya

Potato

Tomato

Brinjal

Bean

Carrot

Capsicum

Bitter_Gourd

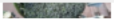
```
train_gen = ImageDataGenerator(rescale = 1.0/255.0) # Normalise the data
train_image_generator = train_gen.flow_from_directory(
                                            train_path,
                                            target_size=(150, 150),
                                            batch_size=32,
                                            class_mode='categorical')


# 2. Validation Set
val_gen = ImageDataGenerator(rescale = 1.0/255.0) # Normalise the data
val_image_generator = train_gen.flow_from_directory(
                                            validation_path,
                                            target_size=(150, 150),
                                            batch_size=32,class_mode='categorical')
```

```
# 3. Test Set
test_gen = ImageDataGenerator(rescale = 1.0/255.0) # Normalise the data
test_image_generator = train_gen.flow_from_directory(
                                            test_path,
                                            target_size=(150, 150),
                                            batch_size=32,
                                            class_mode='categorical')
```

```
Found 1091 images belonging to 15 classes.
Found 650 images belonging to 15 classes.
Found 300 images belonging to 15 classes.
```

```
class_map = dict([(v, k) for k, v in train_image_generator.class_indices.items()])
print(class_map)
```

```
{0: 'Bean', 1: 'Bitter_Gourd', 2: 'Bottle_Gourd', 3: 'Brinjal', 4: 'Broccoli', 5: 'Cabbage', 6: 'Capsicum', 7: 'Carrot', 8: 'Caulif
```



```
model = Sequential() # model object

# Add Layers
model.add(Conv2D(filters=32, kernel_size=3, strides=1, padding='same', activation='relu', input_shape=[150, 150, 3]))
model.add(MaxPooling2D(2, ))
model.add(Conv2D(filters=64, kernel_size=3, strides=1, padding='same', activation='relu'))
model.add(MaxPooling2D(2))

# Flatten the feature map
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(128, activation='relu'))
model.add(Dense(15, activation='softmax'))

# print the model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 150, 150, 32)      896

 max_pooling2d (MaxPooling2  (None, 75, 75, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 75, 75, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 37, 37, 64)        0
 g2D)

 flatten (Flatten)           (None, 87616)             0

 dense (Dense)               (None, 128)               11214976

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 128)               16512

 dense_2 (Dense)             (None, 15)                1935

=================================================================
Total params: 11252815 (42.93 MB)
Trainable params: 11252815 (42.93 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
early_stopping = keras.callbacks.EarlyStopping(patience=5) # Set up callbacks
model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics='accuracy')
hist = model.fit(train_image_generator,
                 epochs=10,
                 verbose=1,
                 validation_data=val_image_generator,
                 steps_per_epoch = 150//32,
                 validation_steps = 30//32,
                 callbacks=early_stopping)
```

```
Epoch 1/10
4/4 [==============================] - ETA: 0s - loss: 3.2986 - accuracy: 0.4531WARNING:tensorflow:Early stopping conditioned on me
4/4 [==============================] - 6s 1s/step - loss: 3.2986 - accuracy: 0.4531
Epoch 2/10
4/4 [==============================] - ETA: 0s - loss: 1.3305 - accuracy: 0.7578WARNING:tensorflow:Early stopping conditioned on me
4/4 [==============================] - 7s 2s/step - loss: 1.3305 - accuracy: 0.7578
Epoch 3/10
4/4 [==============================] - ETA: 0s - loss: 1.4328 - accuracy: 0.7374WARNING:tensorflow:Early stopping conditioned on me
```

```
4/4 [==============================] - 4s 1s/step - loss: 1.4328 - accuracy: 0.7374
Epoch 4/10
4/4 [==============================] - ETA: 0s - loss: 1.3368 - accuracy: 0.7734WARNING:tensorflow:Early stopping conditioned on me
4/4 [==============================] - 6s 1s/step - loss: 1.3368 - accuracy: 0.7734
Epoch 5/10
4/4 [==============================] - ETA: 0s - loss: 1.2822 - accuracy: 0.7578WARNING:tensorflow:Early stopping conditioned on me
4/4 [==============================] - 5s 1s/step - loss: 1.2822 - accuracy: 0.7578
Epoch 6/10
4/4 [==============================] - ETA: 0s - loss: 1.4145 - accuracy: 0.7031WARNING:tensorflow:Early stopping conditioned on me
4/4 [==============================] - 6s 1s/step - loss: 1.4145 - accuracy: 0.7031
Epoch 7/10
4/4 [==============================] - ETA: 0s - loss: 1.2692 - accuracy: 0.7344WARNING:tensorflow:Early stopping conditioned on me
4/4 [==============================] - 5s 1s/step - loss: 1.2692 - accuracy: 0.7344
Epoch 8/10
4/4 [==============================] - ETA: 0s - loss: 1.1875 - accuracy: 0.7422WARNING:tensorflow:Early stopping conditioned on me
4/4 [==============================] - 6s 1s/step - loss: 1.1875 - accuracy: 0.7422
Epoch 9/10
4/4 [==============================] - ETA: 0s - loss: 1.2847 - accuracy: 0.6953WARNING:tensorflow:Early stopping conditioned on me
4/4 [==============================] - 5s 1s/step - loss: 1.2847 - accuracy: 0.6953
Epoch 10/10
4/4 [==============================] - ETA: 0s - loss: 1.0939 - accuracy: 0.7734WARNING:tensorflow:Early stopping conditioned on me
4/4 [==============================] - 6s 1s/step - loss: 1.0939 - accuracy: 0.7734
```

```
model.evaluate(test_image_generator)
```

```
10/10 [==============================] - 100s 11s/step - loss: 3.8680 - accuracy: 0.0667
[3.867973566055298, 0.06666667014360428]
```

```
test_image_path = '../content/drive/MyDrive/Vegetable Images/test/Bean/0001.jpg'

def generate_predictions(test_image_path, actual_label):

    # 1. Load and preprocess the image
    test_img = image.load_img(test_image_path, target_size=(150, 150))
    test_img_arr = image.img_to_array(test_img)/255.0
    test_img_input = test_img_arr.reshape((1, test_img_arr.shape[0], test_img_arr.shape[1], test_img_arr.shape[2]))

    # 2. Make Predictions
    predicted_label = np.argmax(model.predict(test_img_input))
    predicted_vegetable = class_map[predicted_label]
    plt.figure(figsize=(4, 4))
    plt.imshow(test_img_arr)
    plt.title("Predicted Label: {}, Actual Label: {}".format(predicted_vegetable, actual_label))
    plt.grid()
    plt.axis('off')
    plt.show()

# call the function
generate_predictions(test_image_path, actual_label='Beans')
```

```
1/1 [==============================] - 0s 201ms/step
```

Predicted Label: Pumpkin, Actual Label: Beans