In [5]:

```
# example of using a pre-trained model as a classifier
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg16 import decode_predictions
from keras.applications.vgg16 import VGG16
```

In [19]:

```
# load an image from file
image = load_img('download.jpg', target_size=(224, 224))
```

In [20]:

```
# convert the image pixels to a numpy array
image = img_to_array(image)
```

In [21]:

```
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
```

In [22]:

```
# prepare the image for the VGG model
image = preprocess_input(image)
```

In [23]:

```
# load the model
model = VGG16()
```

In [24]:

```
# predict the probability across all output classes
yhat = model.predict(image)
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_function.<locals>.predict_function
at 0x000001F2498996C0> triggered tf.function retracing. Tracing is expensive and the excessive number of tracing
s could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3)
passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2)
, @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to h
ttps://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf
/function for  more details.
1/1 [==============================] - 1s 726ms/step
```

In [25]:

```
# convert the probabilities to class labels
label = decode_predictions(yhat)
# convert the probabilities to class labels
label = decode_predictions(yhat)
```

In [26]:

```
# retrieve the most likely result, e.g. highest probability
label = label[0][0]
```

In [27]:

```
# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))
```

```
castle (34.03%)
```

In [28]:

```
# load an image from file
image = load_img('download2.png', target_size=(224, 224))
```

In [29]:

```
# convert the image pixels to a numpy array
image = img_to_array(image)
```

In [30]:

```
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
```

In [31]:

```
# prepare the image for the VGG model
image = preprocess_input(image)
```

In [32]:

```
# load the model
model = VGG16()
```

In [33]:

```
# predict the probability across all output classes
yhat = model.predict(image)
```

```
WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_function.<locals>.predict_function
at 0x000001F249898860> triggered tf.function retracing. Tracing is expensive and the excessive number of tracing
s could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3)
passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2)
, @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to h
ttps://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf
/function for  more details.
1/1 [==============================] - 1s 1000ms/step
```

In [34]:

```
# convert the probabilities to class labels
label = decode_predictions(yhat)
```

In [35]:

```
# retrieve the most likely result, e.g. highest probability
label = label[0][0]
```

```
label
```

```
Out[35]: ('n09468604', 'valley', 0.448454)
```

In [36]:

```
# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))
```

```
valley (44.85%)
```

In [37]:

```
# load an image from file
image = load_img('download3.jpg', target_size=(224, 224))
```

In [38]:

```
# convert the image pixels to a numpy array
image = img_to_array(image)
```

In [39]:

```
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
```

In [46]:

```
# prepare the image for the VGG model
image = preprocess_input(image)
```

In [47]:

```
# load the model
model = VGG16()
```

In [48]:

```
# predict the probability across all output classes
yhat = model.predict(image)
```

```
1/1 [==============================] - 2s 2s/step
```

In [49]:

```
# convert the probabilities to class labels
label = decode_predictions(yhat)
```

In [50]:

```
# retrieve the most likely result, e.g. highest probability
label = label[0][0]
```

In [51]:

```
# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))
```

```
golden_retriever (77.67%)
```

In [52]:

```
model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_8 (InputLayer)        [(None, 224, 224, 3)]     0         
                                                                 
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0         
                                                                 
 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0         
                                                                 
 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080    
                                                                 
 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080    
                                                                 
 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0         
                                                                 
 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160   
                                                                 
 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808   
                                                                 
 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808   
                                                                 
 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0         
                                                                 
 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808   
                                                                 
 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808   
                                                                 
 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808   
                                                                 
 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0         
                                                                 
 flatten (Flatten)           (None, 25088)             0         
                                                                 
 fc1 (Dense)                 (None, 4096)              102764544 
                                                                 
 fc2 (Dense)                 (None, 4096)              16781312  
                                                                 
 predictions (Dense)         (None, 1000)              4097000   
                                                                 
=================================================================
Total params: 138357544 (527.79 MB)
Trainable params: 138357544 (527.79 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js