```
In [ ]:                           DL Expt-6

In [1]:  # 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
         # load an image from file
         image = load_img('download.jpg', target_size=(224, 224))
         # convert the image pixels to a numpy array
         image = img_to_array(image)
         # reshape data for the model
         image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
         # prepare the image for the VGG model
         image = preprocess_input(image)
         # load the model
         model = VGG16()
         # predict the probability across all output classes
         yhat = model.predict(image)
         # convert the probabilities to class labels
         label = decode_predictions(yhat)
         # retrieve the most likely result, e.g. highest probability
         label = label[0][0]
         # print the classification
         print('%s (%.2f%%)' % (label[1], label[2]*100))
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5
553467904/553467096 [==============================] - 228s 0us/step
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/
imagenet_class_index.json
40960/35363 [==================================] - 0s 0us/step
castle (34.03%)

```
In [5]:  # load an image from file
         image = load_img('download2.png', target_size=(224, 224))
         # convert the image pixels to a numpy array
         image = img_to_array(image)
         # reshape data for the model
         image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
         # prepare the image for the VGG model
         image = preprocess_input(image)
         # load the model
         model = VGG16()
         # predict the probability across all output classes
         yhat = model.predict(image)
         # convert the probabilities to class labels
         label = decode_predictions(yhat)
         # retrieve the most likely result, e.g. highest probability
         label = label[0][0]
         # print the classification
         print('%s (%.2f%%)' % (label[1], label[2]*100))
```

valley (44.85%)

```
In [6]:  # load an image from file
         image = load_img('download3.jpg', target_size=(224, 224))
         # convert the image pixels to a numpy array
         image = img_to_array(image)
         # reshape data for the model
         image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
```

```python
# prepare the image for the VGG model
image = preprocess_input(image)
# load the model
model = VGG16()
# predict the probability across all output classes
yhat = model.predict(image)
# convert the probabilities to class labels
label = decode_predictions(yhat)
# retrieve the most likely result, e.g. highest probability
label = label[0][0]
# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))
```

WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_funct
ion.<locals>.predict_function at 0x0000021D82424EE8> triggered tf.function retraci
ng. Tracing is expensive and the excessive number of tracings could be due to (1)
creating @tf.function repeatedly in a loop, (2) passing tensors with different sha
pes, (3) passing Python objects instead of tensors. For (1), please define your @t
f.function outside of the loop. For (2), @tf.function has experimental_relax_shape
s=True option that relaxes argument shapes that can avoid unnecessary retracing. F
or (3), please refer to https://www.tensorflow.org/tutorials/customization/perform
ance#python_or_tensor_args and https://www.tensorflow.org/api_docs/python/tf/funct
ion for more details.
golden_retriever (84.78%)