In this tutorial, we will focus on the use case of classifying new images using the VGG model.The vgg16 model can be created as follows:

In [2]:
```
from keras.applications.vgg16 import VGG16
model = VGG16()
```

```
Using TensorFlow backend.
```

That’s it.

The first time you run this example, Keras will download the weight files from the Internet and store them in the ~/.keras/models directory.Note that the weights are about 528 megabytes, so the download may take a few minutes depending on the speed of your Internet connection.

The weights are only downloaded once. The next time you run the example, the weights are loaded locally and the model should be ready to use in seconds.

We can use the standard Keras tools for inspecting the model structure

For example, you can print a summary of the network layers as follows:

In [3]:
```
from keras.applications.vgg16 import VGG16
model = VGG16()
print(model.summary())
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (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: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
None
```

The model is huge and by default, the model expects images as input with the size 224 x 224 pixels with 3 channels (e.g. color).

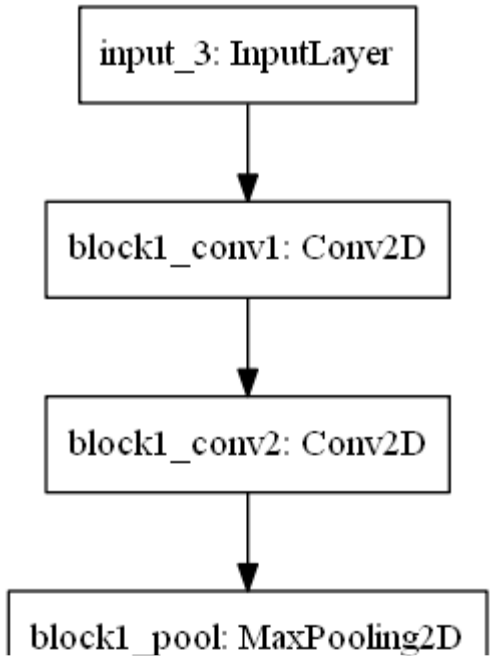
We can also create a plot of the layers in the VGG model, as follows:

In [4]:

```
from keras.applications.vgg16 import VGG16
from keras.utils.vis_utils import plot_model
model = VGG16()
import pydot as pyd
#plot_model(model, to_file='vgg.png')
```

In [5]:
```
plot_model(model, to_file='vgg.png')
```

Out[5]:



The VGG() class takes a few arguments that may only interest you if you are looking to use the model in your own project, e.g. for transfer learning ,that will be helpful for our model

For example:

1)include_top (True): Whether or not to include the output layers for the model. You don't need these if you are fitting the model on your own problem.

2)weights ('imagenet'): What weights to load. You can specify None to not load pre-trained weights if you are interested in training the model yourself from scratch.

3)input_tensor (None): A new input layer if you intend to fit the model on new data of a different size.

4)input_shape (None): The size of images that the model is expected to take if you change the input layer.

5)pooling (None): The type of pooling to use when you are training a new set of output layers.

6)classes (1000): The number of classes (e.g. size of output vector) for the model.

Next, let's look at using the loaded VGG model to classify ad hoc photographs.

Next, let's look at using the loaded VGG model to classify ad hoc photographs.

1. Get a Sample Image

1) First, we need an image we can classify.

In [22]:
```
#our images


import pandas as pd
import time
import wget

a = "https://assets.shop.loblaws.ca/products_jpeg/20004373001/en/20004373001_sml_1_@2x.jpg"
image_url_1 = a
image_jpg = wget.download(image_url_1)
print(image_jpg)
```

```
100% [......................................................................] 19504 / 1950420004373
001_sml_1_@2x.jpg
```

In [ ]:
```

```

2. Load the VGG Model

Load the weights for the VGG-16 model, as we did in the previous section.

In [18]:
```
from keras.applications.vgg16 import VGG16
# load the model
model = VGG16()
```

3. Load and Prepare Image

Next, we can load the image as pixel data and prepare it to be presented to the network.

Keras provides some tools to help with this step.

i)First, we can use the load_img() function to load the image and resize it to the required size of 224×224 pixels

In [24]:
```
from keras.preprocessing.image import load_img
# load an image from file
image = load_img('20004373001_sml_1_@2x.jpg', target_size=(224, 224))
```

In [ ]:
```

```

ii)Next, we can convert the pixels to a NumPy array so that we can work with it in Keras. We can use the img_to_array() function for this.

In [25]:
```
from keras.preprocessing.image import img_to_array
# convert the image pixels to a numpy array
image = img_to_array(image)
```

Note:

The network expects one or more images as input; that means the input array will need to be 4-dimensional: samples, rows, columns, and channels.

We only have one sample (one image). We can reshape the array by calling reshape() and adding the extra dimension.

iii)Reshape data for the model

In [27]:
```
#reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
```

```
Next, the image pixels need to be prepared in the same way as the ImageNet training data was prepared.
Specifically, from the paper:

The only preprocessing we do is subtracting the mean RGB value, computed on the training set, from each pixel.

– Very Deep Convolutional Networks for Large-Scale Image Recognition, 2014.
```

iv) Keras provides a function called preprocess_input() to prepare new input for the network

In [28]:
```
from keras.applications.vgg16 import preprocess_input
# prepare the image for the VGG model
image = preprocess_input(image)
```

Note:

```
We are now ready to make a prediction for our loaded and prepared image.
```

4. Make a Prediction

We can call the predict() function on the model in order to get a prediction of the probability of the image belonging to each of the 1000 known object types.

In [31]:
```
# predict the probability across all output classes
yhat = model.predict(image)
```

5. Interpret Prediction

Keras provides a function to interpret the probabilities called decode_predictions().

It can return a list of classes and their probabilities in case you would like to present the top 3 objects that may be in the photo.

We will just report the first most likely object.

In [32]:
```
from keras.applications.vgg16 import decode_predictions
# 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/download.tensorflow.org/data/imagenet_class_index.json
(https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json)
40960/35363 [==================================] - 0s 1us/step
packet (74.69%)
```

Running the example, we can see that the image is correctly classified as a “coffee mug” with a 75% likelihood.

Summary

In this tutorial, I discovered the VGG 16 convolutional neural network models for image classification.

Specifically, I learned:

About the ImageNet dataset and competition and the VGG winning models. How to load the VGG model in Keras and summarize its structure. How to use the loaded VGG model to classifying objects in ad hoc photographs.

Refernce:

https://machinelearningmastery.com/use-pre-trained-vgg-model-classify-objects-photographs/ (https://machinelearningmastery.com/use-pre-trained-vgg-model-classify-objects-photographs/)

In [ ]: