Convolutional Neural Networks

In your upcoming project, you will download pre-computed bottleneck features. In this notebook, we'll show you how to calculate VGG-16 bottleneck features on a toy dataset. Note that unless you have a powerful GPU, computing the bottleneck features takes a significant amount of time.

1. Load and Preprocess Sample Images

Before supplying an image to a pre-trained network in Keras, there are some required preprocessing steps. You will learn more about this in the project; for now, we have implemented this functionality for you in the first code cell of the notebook. We have imported a very small dataset of 8 images and stored the preprocessed image input as `img_input`. Note that the dimensionality of this array is `(8, 224, 224, 3)`. In this case, each of the 8 images is a 3D tensor, with shape `(224, 224, 3)`.

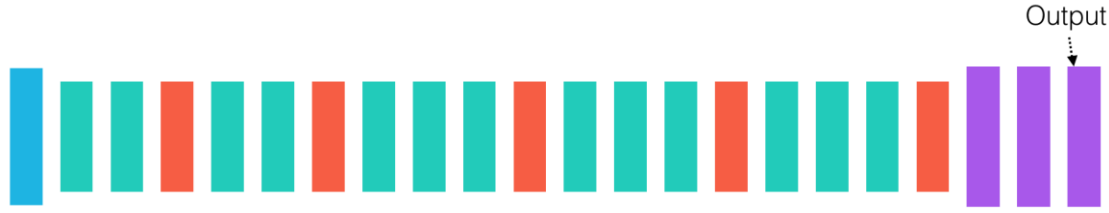
In [1]:

```
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing import image
import numpy as np
import glob

img_paths = glob.glob("images/*.jpg")

def path_to_tensor(img_path):
    # loads RGB image as PIL.Image.Image type
    img = image.load_img(img_path, target_size=(224, 224))
    # convert PIL.Image.Image type to 3D tensor with shape (224, 224, 3)
    x = image.img_to_array(img)
    # convert 3D tensor to 4D tensor with shape (1, 224, 224, 3) and return 4D tensor
    return np.expand_dims(x, axis=0)

def paths_to_tensor(img_paths):
    list_of_tensors = [path_to_tensor(img_path) for img_path in img_paths]
    return np.vstack(list_of_tensors)

# calculate the image input. you will learn more about how this works the project!
img_input = preprocess_input(paths_to_tensor(img_paths))

print(img_input.shape)
```

```
Using TensorFlow backend.
(8, 224, 224, 3)
```

2. Recap How to Import VGG-16

Recall how we import the VGG-16 network (including the final classification layer) that has been pre-trained on ImageNet



.

In [2]:

```
from keras.applications.vgg16 import VGG16
model = VGG16()
model.summary()
```

Layer (type)	Output Shape	Param #
input_1 (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: 138,357,544.0
Trainable params: 138,357,544.0
Non-trainable params: 0.0

For this network, `model.predict` returns a 1000-dimensional probability vector containing the predicted probability that an image returns each of the 1000 ImageNet categories. The dimensionality of the obtained output from passing `img_input`through the model is `(8, 1000)`. The first value of `8` merely denotes that 8 images were passed through the network.

In [3]:

```
model.predict(img_input).shape
```

Out[3]:

```
(8, 1000)
```

3. Import the VGG-16 Model, with the Final Fully-Connected Layers Removed

When performing transfer learning, we need to remove the final layers of the network, as they are too specific to the ImageNet database. This is accomplished in the code cell below.

In [4]:

```
from keras.applications.vgg16 import VGG16
model = VGG16(include_top=False)
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, None, None, 3)     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, None, None, 64)    1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, None, None, 64)    36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, None, None, 64)    0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, None, None, 128)   73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, None, None, 128)   147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, None, None, 128)   0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, None, None, 256)   295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, None, None, 256)   590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, None, None, 256)   590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, None, None, 256)   0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, None, None, 512)   1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
```

```
block4_conv3 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, None, None, 512)   0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, None, None, 512)   0
=================================================================
Total params: 14,714,688.0
Trainable params: 14,714,688.0
Non-trainable params: 0.0
_________________________________________________________________
```

4. Extract Output of Final Max Pooling Layer

Now, the network stored in `model` is a truncated version of the VGG-16 network, where the final three fully-connected layers have been removed. In this case, `model.predict` returns a 3D array (with dimensions 7×7×5127×7×512) corresponding to the final max pooling layer of VGG-16. The dimensionality of the obtained output from passing `img_input` through the model is `(8, 7, 7, 512)`. The first value of `8` merely denotes that 8 images were passed through the network.

In [5]:

```
print(model.predict(img_input).shape)
```

```
(8, 7, 7, 512)
```

This is exactly how we calculate the bottleneck features for your project!