Part 4b - Loading and Benchmarking

In this notebook we will cover the following topics:

- Load a trained model from disk
- Benchmark the performance of the model on the CPU and GPU

In [1]:
```
import numpy as np
np.warnings.filterwarnings('ignore')  # Hide np.floating warning

import keras

from keras.datasets import cifar10

# Prevent TensorFlow from grabbing all the GPU memory
import tensorflow as tf
config = tf.ConfigProto()
config.gpu_options.allow_growth=True
sess = tf.Session(config=config)

import holoviews as hv
hv.extension('bokeh')
```

```
Using TensorFlow backend.
```

Load the Data

In [2]:
```
from keras.datasets import cifar10
import keras.utils

(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Save an unmodified copy of y_test for later, flattened to one column
y_test_true = y_test[:,0].copy()

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# The data only has numeric categories so we also have the string labels bel
ow
cifar10_labels = np.array(['airplane', 'automobile', 'bird', 'cat', 'deer',
                           'dog', 'frog', 'horse', 'ship', 'truck'])
```

Load the Model

Keras provides a `load_model()` function which recreates the model in memory from the provided HDF5 file:

In [3]:
```
from keras.models import load_model
```

This can take a few seconds depending on how large the model is:

In [4]:
```
%%time
gpu_model = load_model('cifar10_model.hdf5')
```

```
CPU times: user 813 ms, sys: 138 ms, total: 951 ms
Wall time: 952 ms
```

In [5]:
```
gpu_model.predict_classes(x_test)
```

Out[5]:
```
array([3, 8, 8, ..., 5, 1, 7])
```

Benchmark CPU vs. GPU

While the acceleration provided by the GPU is very valuable for training, it is not always necessary when using a trained model for prediction only.

Making this tradeoff depends on:

- Whether your deployment target has GPUs
- What are your throughput and latency requirements?
- Can you batch prediction requests, or do you have to process them one at a time?

To see how the CPU and GPU performance of our model compares in this notebook, we need to trick TensorFlow into running the model on the CPU even though a GPU is present. TensorFlow provides a context manager `tf.device()` which can be used to control where operations happen. If we load the model while the TensorFlow is pinning operations to the CPU device, our model will always run on the CPU:

In [6]:
```
with tf.device("/device:CPU:0"):
    cpu_model = load_model('cifar10_model.hdf5')
    print(cpu_model.predict_classes(x_test))
```
```
[3 8 8 ... 5 1 7]
```

And now we can compare CPU and GPU performance on 10000 test images (the ideal case for the GPU):

In [7]:
```
print('GPU performance: %d images' % x_test.shape[0])
%timeit gpu_model.predict_classes(x_test)
print('CPU performance: %d images' % x_test.shape[0])
%timeit cpu_model.predict_classes(x_test)
```
```
GPU performance: 10000 images
1.25 s ± 5.52 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
CPU performance: 10000 images
23 s ± 70 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Depending on your GPU and CPU, this spread can be 10x or more.

Now let's look at the performance for one image:

In [8]:
```
print('GPU performance: 1 image')
%timeit gpu_model.predict_classes(x_test[:1])
print('CPU performance: 1 image')
%timeit cpu_model.predict_classes(x_test[:1])
```
```
GPU performance: 1 image
1.89 ms ± 15.5 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
CPU performance: 1 image
3.55 ms ± 67.1 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

The spread is much smaller, indicating that the CPU might be sufficient for models which have to process inputs one at a time. (For example, a realtime classifier.)

Experiments to Try

- Try going back to notebook #3, saving the more complex model to a different file, and loading it here.
  How do the CPU and GPU performance compare?

If you screw everything up, you can use File / Revert to Checkpoint to go back to the first version of the notebook and restart the Jupyter kernel with Kernel / Restart.

In [ ]:
```
https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle
&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkSh
ape=4,2&seed=0.95025&showTestData=false&discretize=false&percTrainData=50&x=tr
ue&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&co
sY=false&sinY=false&collectStats=false&problem=classification&initZero=false&h
ideText=false
( great examples for tensorflow with GPU)
```