Tensorflow Image Recognition Tutorial

This tutorial shows how we can use MLDB's TensorFlow (https://www.tensorflow.org) integration to do image recognition. TensorFlow is Google's open source deep learning library.

We will load the Inception-v3 model (http://arxiv.org/abs/1512.00567) to generate descriptive labels for an image. The *Inception* model is a deep convolutional neural network and was trained on the ImageNet (http://image-net.org) Large Visual Recognition Challenge dataset, where the task was to classify images into 1000 classes.

To offer context and a basis for comparison, this notebook is inspired by TensorFlow's Image Recognition tutorial (https://www.tensorflow.org/versions/r0.7/tutorials/image_recognition/index.html).

Initializing pymldb and other imports

The notebook cells below use pymldb's Connection class to make REST API (../../../../doc/#builtin/WorkingWithRest.md.html) calls. You can check out the Using pymldb Tutorial (../../../../doc/nblink.html#_tutorials/Using pymldb Tutorial) for more details.

In [1]:
```
from pymldb import Connection
mldb = Connection()
```

Loading a TensorFlow graph

To load a pre-trained TensorFlow graphs in MLDB, we use the tensorflow.graph function type (../../../../doc/#/v1/plugins/tensorflow/doc/TensorflowGraph.md.html).

Below, we start by creating two functions. First, the fetcher function allows us to fetch a binary blob from a remote URL. Second, the inception function that will be used to execute the trained network and that we parameterize in the following way:

- **modelFileUrl**: Path to the Inception-v3 model file. The archive prefix and # separator allow us to load a file inside a zip archive. (more details (../../../../doc/#builtin/Url.md.html))
- **input**: As input to the graph, we provide the output of the fetch function called with the url parameter. When we call it later on, the image located at the specified URL will be downloaded and passed to the graph.
- **output**: This specifies the layer from which to return the values. The softmax layer is the last layer in the network so we specify that one.

In [2]:
```
inceptionUrl = 'http://public.mldb.ai/models/inception_dec_2015.zip'

print mldb.put('/v1/functions/fetch', {
   "type": 'fetcher',
   "params": {}
})

print mldb.put('/v1/functions/inception', {
   "type": 'tensorflow.graph',
   "params": {
      "modelFileUrl": 'archive+' + inceptionUrl + '#tensorflow_inception_graph.pb',
      "inputs": 'fetch({url})[content] AS "DecodeJpeg/contents"',
      "outputs": "softmax"
   }
})
```

```
<Response [201]>
<Response [201]>
```

Scoring an image

To demonstrate how to run the network on an image, we re-use the same image as in the Tensorflow tutorial, the picture of Admiral Grace Hopper (https://en.wikipedia.org/wiki/Grace_Hopper):

<img src="https://www.tensorflow.org/versions/r0.7/images/grace_hopper.jpg (https://www.tensorflow.org/versions/r0.7/images/grace_hopper.jpg)" width=350>

The following query applies the inception function on the URL of her picture:

In [3]:
```
amazingGrace = "https://www.tensorflow.org/versions/r0.7/images/grace_hopper.jpg"

mldb.query("SELECT inception({url: '%s'}) as *" % amazingGrace)
```

Out[3]:

	softmax.0.0	softmax.0.1	softmax.0.2	softmax.0.3	softmax.0.4	softmax.0.5	softmax.0.6	softmax.0.7	softmax.0.8	softma
_rowName										
result	0.000067	0.000032	0.000055	0.000036	0.000047	0.000047	0.00002	0.000045	0.00006	0.0000

1 rows × 1008 columns

This is great! With only 3 REST calls we were able to run a deep neural network on an arbitrary image off the internet.

Inception as a real-time endpoint

Not only is this function available in SQL queries within MLDB, but as all MLDB functions, it is also available as a REST endpoint. This means that when we created the inception function above, we essentially created an real-time API running the *Inception* model that any external service or device can call to get predictions back.

The following REST call demonstrates how this looks:

In [4]:
```
result = mldb.get('/v1/functions/inception/application', input={"url": amazingGrace})

print result.url + '\n\n' + repr(result) + '\n'

import numpy as np
print "Shape:"
print np.array(result.json()["output"]["softmax"]["val"]).shape
```

```
http://localhost/v1/functions/inception/application?input=%7B%22url%22%3A+%22https%3A%2F%2Fwww.tensorflow.org%2Fversions%2Fr0.7%2Fimages%2Fgrace_hopper.jpg%22%7D

<Response [200]>

Shape:
(1, 1008)
```

Interpreting the prediction

Running the network gives us a 1008-dimensional vector. This is because the network was originally trained on the Image net categories and we created the inception function to return the *softmax* layer which is the output of the model.

To allow us to interpret the predictions the network makes, we can import the ImageNet labels in an MLDB dataset like this:

In [5]:
```
print mldb.put("/v1/procedures/imagenet_labels_importer", {
    "type": "import.text",
    "params": {
        "dataFileUrl": 'archive+' + inceptionUrl + '#imagenet_comp_graph_label_strings.txt',
        "outputDataset": {"id": "imagenet_labels", "type": "sparse.mutable"},
        "headers": ["label"],
        "named": "lineNumber() -1",
        "offset": 1,
        "runOnCreation": True
    }
})
```

<Response [201]>

The contents of the dataset look like this:

In [6]:
```
mldb.query("SELECT * FROM imagenet_labels LIMIT 5")
```

Out[6]:

	label
_rowName	
822	soup bowl
471	green lizard
77	Irish wolfhound
821	dishrag
569	swing

The labels line up with the *softmax* layer that we extract from the network. By joining the output of the network with the imagenet_labels dataset, we can essentially label the output of the network.

The following query scores the image just as before, but then transposes the output and then joins the result to the labels dataset. We then sort on the score to keep only the 10 highest values:

In [7]:
```
mldb.query("""
   SELECT scores.pred as score
   NAMED imagenet_labels.label
   FROM transpose(
      (
         SELECT flatten(inception({url: '%s'})[softmax]) as *
         NAMED 'pred'
      )
   ) AS scores

   LEFT JOIN imagenet_labels ON
      imagenet_labels.rowName() = scores.rowName()

   ORDER BY score DESC
   LIMIT 10
""" % amazingGrace)
```

Out[7]:

	score
_rowName	
military uniform	0.808008
suit	0.022845
academic gown	0.009540
bearskin	0.009413
pickelhaube	0.007743
bolo tie	0.006794
bulletproof vest	0.005836
bow tie	0.004035
cornet	0.003984
Windsor tie	0.003208

Where to next?

You can now look at the Transfer Learning with Tensorflow (../../../../doc/nblink.html#_demos/Transfer Learning with Tensorflow) demo.

In [8]: