mldb.ai

(/)

This page is part of the documentation for the Machine Learning Database (http://mldb.ai).

It is a static snapshot of a Notebook which you can play with interactively by trying MLDB online now (/doc/#builtin/Running.md.html).
It's free and takes 30 seconds to get going.

Transfer Learning on Images with Tensorflow

This demo will demonstrate how to do transfer learning to leverage the power of a deep convolutional neural network without having to train one yourself. Most people do not train those types of networks from scratch because of the large data and computational power requirements. What is more common is to train the network on a large dataset (unrelated to our task) and then leverage the representation it learnt in one of the following ways:

- as the initialization of a new network which will then be trained on our specific task (fine tuning)
- as a feature generator, essentially passing in our examples in the network so it can embed them in the abstract representation it learnt

This notebook will be doing the latter using the Inception-v3 model (http://arxiv.org/abs/1512.00567) that was trained on the ImageNet (http://image-net.org) Large Visual Recognition Challenge dataset made up of over 1 million images, where the task was to classify images into 1000 classes.

We will cover three topics in this demo:

- Unsupervised learning: using the image embedding as input in a dimensionality reduction algorithm for visualization
- Supervised learning: using the image embedding as features for a multi-class classification task
- The *DeepTeach* plugin: showing a plugin that uses the techniques introduced to build a binary image classifier by doing similarity search through a web UI

The following videos shows *DeepTeach* in action:

In [1]:
```
from IPython.display import YouTubeVideo
YouTubeVideo('7hZ3X37Qwc4')
```

Out[1]:

We recommend reading the Tensorflow Image Recognition Tutorial (../../../../doc/nblink.html#_tutorials/Tensorflow Image Recognition Tutorial) before going though this demo.

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 [2]:
```
from pymldb import Connection
mldb = Connection()
import urllib2, pandas as pd, numpy as np, matplotlib.pyplot as plt
from matplotlib.offsetbox import TextArea, DrawingArea, OffsetImage, AnnotationBbox
from matplotlib._png import read_png
%matplotlib inline
```

The dataset

In the Tensorflow Image Recognition Tutorial (../../../../doc/nblink.html#_tutorials/Tensorflow Image Recognition Tutorial) tutorial, you saw how to embed the image of Admiral Grace Hopper using the *Inception* model. To embed a whole dataset of images, we do the same thing but within a procedure of type transform. We will not cover this in details in the notebook, but the detailed code is available in the *Dataset Builder* plugin's code (https://github.com/mldbai/dataset-builder/blob/master/routes.py#L317).

We ship the plugin with 4 pre-assembled datasets: real-estate, recipes, transportation and pets. We start by loading the embeddings of the images for the real-estate dataset:

In [3]:
```
prefix = "http://public.mldb.ai/datasets/dataset-builder"

print mldb.put("/v1/procedures/embedded_images", {
    "type": "import.text",
    "params": {
        "dataFileUrl": prefix + "/cache/dataset_creator_embedding_realestate.csv.gz",
        "outputDataset": {
                "id": "embedded_images_realestate",
                "type": "embedding"
            },
        "select": "* EXCLUDING(rowName)",
        "named": "rowName",
        "runOnCreation": True
    }
})
```

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

The dataset we just imported has one row per image and the dense columns are the 2048-dimensional embeddings. We used the second to last layer of the network, the pool_3 layer, which is less specialized than the final softmax layer of the network. Since the *Inception* model was trained on the ImageNet task, the last layer has been trained to perform very well on that specific task, while the previous layers are more abstract representations and are more suitable for transfer learning tasks.

The following query shows the embedding values for 2 rows:

In [4]:
```
mldb.query("SELECT * FROM embedded_images_realestate ORDER BY rowHash() ASC LIMIT 5")
```

Out[4]:

	"pool_3.0.0.0.0"	"pool_3.0.0.0.1"	"pool_3.0.0.0.2"	"pool_3.0.0.0.3"	"pool_3.0.0.0.4"	"pool_3.0.0.0.5"	"pool_3.0.0.0.6
_rowName							
condo-13	0.037069	0.290698	0.332847	0.322052	0.505879	0.187525	0.023030
office_building-11	0.003208	0.231195	0.231483	0.056335	0.183423	0.137407	0.432620
igloo-16	0.212397	0.224847	0.085570	0.018010	0.127204	0.020553	0.252013
sand_castle-9	0.113618	0.571280	0.026013	0.190822	0.059604	0.249640	0.296866
castle-18	0.140019	0.237392	0.059138	0.130825	0.143036	0.065903	0.073389

5 rows × 2048 columns

The real-estate dataset contains images of different types of buildings. The following query shows the different categories:

In [5]:
```
mldb.query("""
    SELECT count(*) as count
    FROM embedded_images_realestate
    GROUP BY regex_replace(rowName(), '-[\\d]+', '')
""")
```

Out[5]:

	count
_rowName	
"[""beach_house""]"	20
"[""cabin""]"	20
"[""castle""]"	20
"[""condo""]"	20
"[""condo_building""]"	20
"[""cottage""]"	20
"[""duplex""]"	20
"[""hut""]"	20
"[""igloo""]"	20
"[""monument""]"	20
"[""office_building""]"	20
"[""sand_castle""]"	20
"[""suburban_house""]"	20
"[""town_house""]"	20
"[""triplex""]"	20

Here are a few sample images:

condo-13		sand_castle-10	
office_building-11		town_house-2	

Unsupervised learning

The first transfer learning task we will do is using the rich abstract embedding of the images to run an unsupervised dimensionality reduction algorithm to visualize the real-estate dataset.

In the following query, we use the t-SNE algorithm (../../../../doc/#builtin/procedures/TsneProcedure.md.html) to do dimensionality reduction for our visualization:

In [6]:
```
print mldb.put("/v1/procedures/tsne", {
    "type": "tsne.train",
    "params": {
        "trainingData": "SELECT * FROM embedded_images_realestate",
        "rowOutputDataset": "tsne_embedding",
        "numOutputDimensions": 2,
        "runOnCreation": True
    }
})
```

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

The tsne_embedding dataset that the t-SNE procedure generated gives us x and y coordinates for all our images:

In [7]:
```
mldb.query("SELECT * from tsne_embedding limit 2")
```

Out[7]:

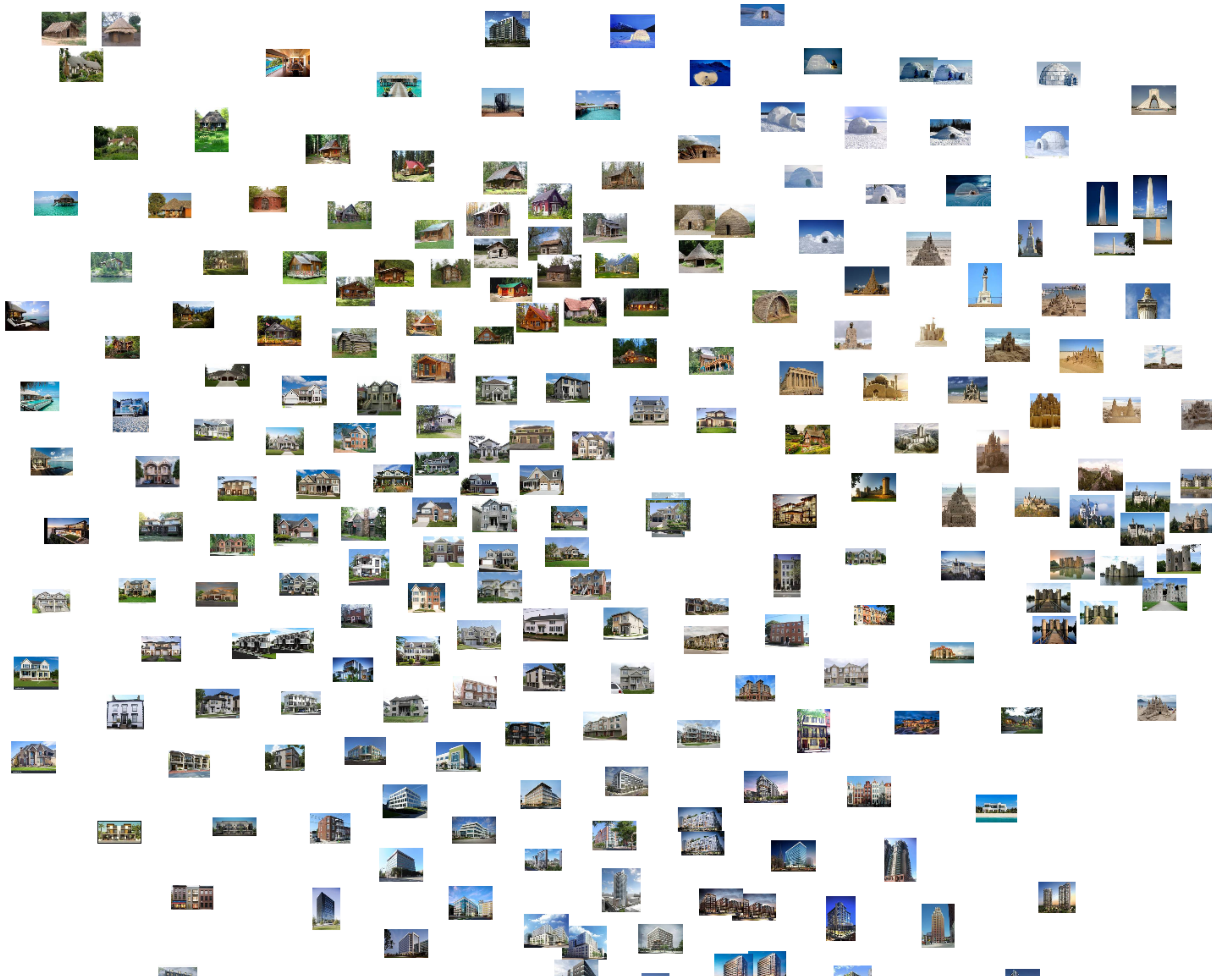
	x	y
_rowName		
condo-13	78.710686	-223.040863
office_building-11	-41.110279	-99.719650

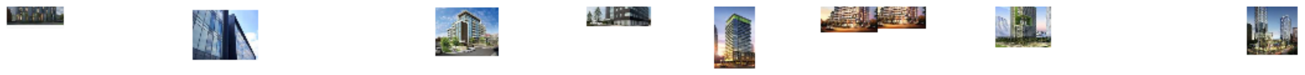
We can now create a scatter plot of all the images in our dataset, positioning them at the coordinates provided by the t-SNE algorithm:

In [8]:
```
image_prefix = "http://public.mldb.ai/datasets/dataset-builder/images/realestate_png/"

df = mldb.query("SELECT * from tsne_embedding")
bounds = df.quantile([.05, .95]).T.values
fig = plt.figure(figsize=(18, 15), frameon=False)
ax = fig.add_subplot(111, xlim=bounds[0], ylim=bounds[1])
plt.axis('off')

for x in df.iterrows():
    imagebox = OffsetImage(read_png(urllib2.urlopen(image_prefix + "%s.png" % x[0])), zoom=0.35)
    ax.add_artist(AnnotationBbox(imagebox, xy=(x[1]["x"], x[1]["y"]), xycoords='data', frameon=False))
```





We can clearly see clusters of similar images in the scatter plot above. Most of the work to make this possible was already done, when the convolutional network was trained. This allowed a simple dimensionality reduction algorithm to get decent results.

Supervised learning

The second transfer learning task we will do is to use the image embeddings as features for a supervised multi-class classifier. In this example, we will train a bagged boosted decision tree to do multi-class classification on a subset of the labels present in the real-estate dataset used in the *unsupervised learning* example.

The following call will generate our training dataset, keeping only the images that are either a castle, an igloo, a cabin, a town house, a condo or a sand castle:

In [9]:
```
print mldb.put("/v1/procedures/<id>", {
    "type": "transform",
    "params": {
        "inputData": """
            SELECT *, regex_replace(rowName(), '-[\\d]+', '') as category
            FROM embedded_images_realestate
            WHERE regex_replace(rowName(), '-[\\d]+', '') IN
                        ('castle', 'igloo', 'cabin', 'town_house', 'condo', 'sand_castle')
        """,
        "outputDataset": "training_dataset",
        "runOnCreation": True
    }
})
```

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

We can now take look at the generated dataset:

In [10]:
```
mldb.query("select * from training_dataset limit 3")
```

Out[10]:

	category	**"pool_3.0.0.0.0"**	**"pool_3.0.0.0.1"**	**"pool_3.0.0.0.2"**	**"pool_3.0.0.0.3"**	**"pool_3.0.0.0.4"**	**"pool_3.0.0.0.5"**	**"poo**
_rowName								
cabin-2	cabin	0.119983	0.497521	0.080315	0.010898	0.252862	0.035614	0.054
town_house-18	town_house	0.086656	0.216155	0.058279	0.042195	0.335481	0.101453	0.525
castle-11	castle	0.146533	0.308326	0.067183	0.166971	0.297405	0.061250	0.056

3 rows × 2049 columns

We can now use a procedure of type classifier.experiment (../../../../doc/#builtin/procedures/ExperimentProcedure.md.html) to train and test our classifier. We will be using 2/3 of our data for training and the rest for testing:

In [11]:
```
rez = mldb.put("/v1/procedures/<id>", {
    "type": "classifier.experiment",
    "params": {
        "experimentName": "realestate",
        "inputData": """
            SELECT
                {* EXCLUDING(category)} as features,
                category as label
            FROM training_dataset
        """,
        "datasetFolds": [
                {
                    "trainingWhere": "rowHash() % 3 < 2",
                    "testingWhere": "rowHash() % 3 = 2",
                }
            ],
        "modelFileUrlPattern": "file:///mldb_data/realestate.cls",
        "algorithm": "bbdt",
        "mode": "categorical",
        "outputAccuracyDataset": True,
        "runOnCreation": True
    }
})

runResults = rez.json()["status"]["firstRun"]["status"]["folds"][0]["resultsTest"]
print rez
```

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

We can now look at the average classification accuracy for all labels:

In [12]:
```
pd.DataFrame.from_dict({"weightedStatistics": runResults["weightedStatistics"]})
```

Out[12]:

	weightedStatistics
accuracy	0.943985
f1Score	0.840559
precision	0.844481
recall	0.848485
support	33.000000

Those results are pretty good, and again this shows how good the features we are getting from the ConvNet are.

We can look at the confusion matrix to give us a better idea of where the mistakes were made:

In [13]:
```
pd.DataFrame(runResults["confusionMatrix"]).pivot_table(index="actual", columns="predicted", fill_value=0)
```

Out[13]:

	count					
predicted	**cabin**	**castle**	**condo**	**igloo**	**sand_castle**	**town_house**
actual						
cabin	6	0	0	0	0	0
castle	0	4	0	0	0	0
condo	0	0	3	0	1	1
igloo	0	0	0	4	0	0
sand_castle	0	0	0	0	7	0
town_house	1	0	2	0	0	4

The *DeepTeach* plugin

DeepTeach is a concrete example of the powerful things we can do with transfer learning. It essentially does similarity search to allow a user to very quickly build a binary image classifier by doing a combination of thow things:

- nearest neighbour search by computing the distance between the embeddings of different images
- supervised classifier training using the embeddings as features

You can see the plugin in action in the video included at the top of the notebook, or read the blog post (http://blog.mldb.ai/blog/posts/2016/10/deepteach/) explaining how it works.

Installing the plugin

To play with the plugin yourself, you can install it in your MLDB instance by simply running the next cell:

In [14]:
```
print mldb.put('/v1/plugins/deepteach', {
    "type": "python",
    "params": {
        "address": "git://github.com/mldbai/deepteach.git"
    }
})
```

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

If you did not change the name of the plugin in the cell above, you can load the plugin's UI here (../../../../v1/plugins/deepteach/routes/static/index.html).

NOTE: this only works if you're running this Notebook live, not if you're looking at a static copy on http://docs.mldb.ai (http://docs.mldb.ai). See the documentation for Running MLDB (../../../../doc/#builtin/Running.md.html).

See also

- CNN Features off-the-shelf: an Astounding Baseline for Recognition (http://arxiv.org/abs/1403.6382)
- How transferable are features in deep neural networks? (http://arxiv.org/abs/1411.1792)