jupyter DeepLearningImageMatch Last Checkpoint: a few seconds ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help | Python [conda env:gl-env]

Code CellToolbar

Let's use deep learning to find out similar images.

author: bhavesh patel

we will use K nearest neighbour algorithm.

In [1]:
```
import graphlab
```

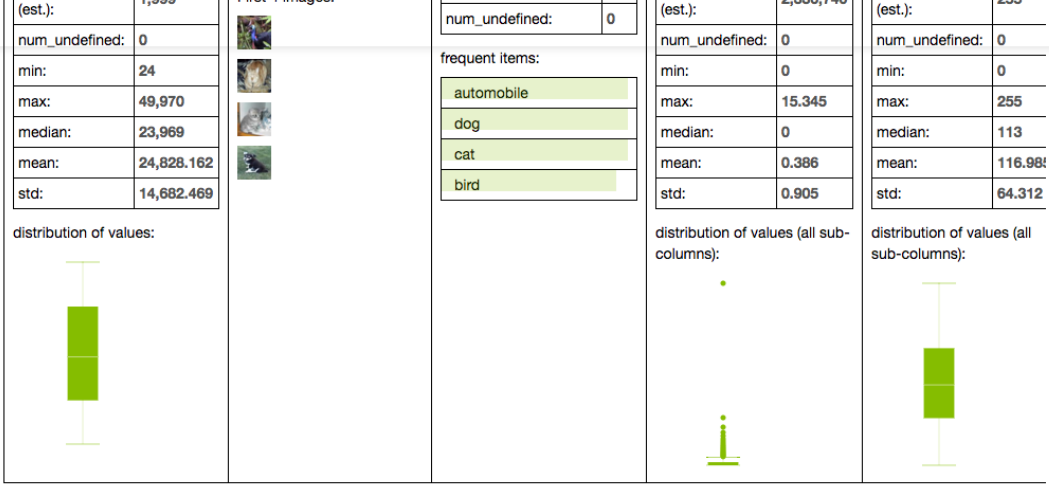
In [2]:
```
# Limit number of worker processes. This preserves system memory, which prevents hosted notebooks from crashing.
graphlab.set_runtime_config('GRAPHLAB_DEFAULT_NUM_PYLAMBDA_WORKERS', 4)
```

```
[INFO] graphlab.cython.cy_server: GraphLab Create v2.1 started. Logging: /tmp/graphlab_server_1481433924.log

This non-commercial license of GraphLab Create for academic use is assigned to bhaveshhk8@gmail.com and will expire o
n October 17, 2017.
```

In [9]:
```
# load the data for images.

image_train = graphlab.SFrame('image_train_data/')
```

In [5]:
```
# view data.
graphlab.canvas.set_target('ipynb')
image_train.show()
```

id		image		label		deep_features		image_array	
dtype:	int	dtype:	Image	dtype:	str	dtype:	array	dtype:	array
num_unique (est.):	1,999	First 4 images:		num_unique (est.):	4	num_unique (est.):	2,336,740	num_unique (est.):	255
num_undefined:	0			num_undefined:	0	num_undefined:	0	num_undefined:	0
min:	24			frequent items:		min:	0	min:	0
max:	49,970			automobile		max:	15.345	max:	255
median:	23,969			dog		median:	0	median:	113
mean:	24,828.162			cat		mean:	0.386	mean:	116.985
std:	14,682.469			bird		std:	0.905	std:	64.312
distribution of values:						distribution of values (all sub-columns):		distribution of values (all sub-columns):	



In [8]:
```
# trian the model using nearest-neighbour algorithm.

knn_model = graphlab.nearest_neighbors.create(image_train,
                                              features=['deep_features'],
                                              label='id')
```

```
Starting brute force nearest neighbors model training.
```

In [18]:
```
# now let's pick 4th image as its cat as we show in above show() command.

test1=image_train[3:4]
test1['image'].show()
```

All 1 images in *<SArray>*

In [19]:
```
# now let's find out similar images.
knn_model.query(test1)
```

```
Starting pairwise querying.

+--------------+---------+-------------+--------------+

| Query points | # Pairs | % Complete. | Elapsed Time |

+--------------+---------+-------------+--------------+
```

```
| 0              | 1        | 0.0498753    | 19.527ms      |
| Done           |          | 100          | 251.811ms     |
+---------------+---------+-------------+--------------+
```

Out[19]:

query_label	reference_label	distance	rank
0	70	0.0	1
0	19437	37.563553912	2
0	41989	40.4601125879	3
0	40249	40.9408480727	4
0	11000	41.3051067894	5

[5 rows x 4 columns]

In [20]:

```
# we don't see the images, so that's not easy to understand.

res = knn_model.query(test1)
```

```
Starting pairwise querying.
+---------------+---------+-------------+--------------+
| Query points | # Pairs | % Complete. | Elapsed Time |
+---------------+---------+-------------+--------------+
| 0              | 1        | 0.0498753    | 33.866ms      |
| Done           |          | 100          | 257.816ms     |
+---------------+---------+-------------+--------------+
```

In [21]:

```
# now let's retrive the image.
image_train.filter_by(res['reference_label'],'id').show()
```

id

dtype:	int
num_unique (est.):	5
num_undefined:	0
min:	70
max:	41,989
median:	19,437
mean:	22,549
std:	16,368.097

distribution of values:

image

dtype:	Image

First 4 images:

label

dtype:	str
num_unique (est.):	3
num_undefined:	0

frequent items:

dog
cat
automobile

deep_features

dtype:	array
num_unique (est.):	6,762
num_undefined:	0
min:	0
max:	11.237
median:	0
mean:	0.424
std:	0.933

distribution of values (all sub-columns):

image_array

dtype:	array
num_unique (est.):	249
num_undefined:	0
min:	0
max:	255
median:	109
mean:	108.864
std:	61.72

distribution of values (all sub-columns):

In [25]:

```
# good results, but not perfect.
# let's create a function to make it easier to find neighbours and find out the quality of results.

def show_neighbours(i):
    res = knn_model.query(image_train[i:i+1])
    fil_res = image_train.filter_by(res['reference_label'],'id')
    fil_res['image'].show()
```

In [26]:

```
show_neighbours(13)
```

```
Starting pairwise querying.
+---------------+---------+-------------+--------------+
| Query points | # Pairs | % Complete. | Elapsed Time |
+---------------+---------+-------------+--------------+
| 0              | 1        | 0.0498753    | 24.224ms      |
| Done           |          | 100          | 218.68ms      |
+---------------+---------+-------------+--------------+
```

All 5 images in *<SArray>*

In [ ]:

```
# nice.  First image is the query and it is matching with all cars with almost similar white color.
```

```
# let's play bit more.
```

In [29]:
```
show_neighbours(1537)
```

```
Starting pairwise querying.

+--------------+---------+-------------+--------------+

| Query points | # Pairs | % Complete. | Elapsed Time |

+--------------+---------+-------------+--------------+

| 0            | 1       | 0.0498753   | 25.668ms     |

| Done         |         | 100         | 233.316ms    |

+--------------+---------+-------------+--------------+
```

All 5 images in *<SArray>*

In [ ]:
```
# for car, the results are coming out good.  For rest it has good success.
```