jupyter DeepLearningForCategorizedImages Last Checkpoint: a minute ago (autosaved)

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

Code CellToolbar

Deep Learning for image retrieval for a given category

author: bhavesh patel

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)
```

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

```
[INFO] graphlab.cython.cy_server: GraphLab Create v2.1 started. Logging: /tmp/graphlab_server_1481438263.log
```

In [3]:
```
# load data.

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

In [4]:
```
# view data.

graphlab.canvas.set_target('ipynb')
image_train.show()
```

id	
dtype:	int
num_unique (est.):	1,999
num_undefined:	0
min:	24
max:	49,970
median:	23,969
mean:	24,828.162
std:	14,682.469

distribution of values:

image	
dtype:	Image

First 4 images:

label	
dtype:	str
num_unique (est.):	4
num_undefined:	0

frequent items:

automobile
dog
cat
bird

deep_features	
dtype:	array
num_unique (est.):	2,336,740
num_undefined:	0
min:	0
max:	15.345
median:	0
mean:	0.386
std:	0.905

distribution of values (all sub-columns):

image_array	
dtype:	array
num_unique (est.):	255
num_undefined:	0
min:	0
max:	255
median:	113
mean:	116.985
std:	64.312

distribution of values (all sub-columns):

In [6]:
```
image_train['label'].sketch_summary()
```

Out[6]:

```
+------------------+-------+----------+
|       item       | value | is exact |
+------------------+-------+----------+
|      Length      |  2005 |   Yes    |
| # Missing Values |   0   |   Yes    |
| # unique values  |   4   |    No    |
+------------------+-------+----------+

Most frequent items:
+-------+------------+-----+-----+------+
| value | automobile | cat | dog | bird |
+-------+------------+-----+-----+------+
| count |    509     | 509 | 509 | 478  |
+-------+------------+-----+-----+------+
```

In [7]:
```
# there are total 2005 records, with 509 autos, 509 cats, 509 dogs and 478 birds.
```

In [8]:
```
dog_data = image_train[image_train['label'] == 'dog']
```

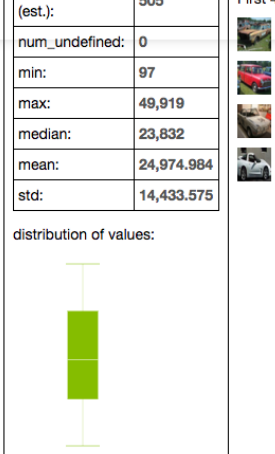
In [13]:
```
auto_data = image_train[image_train['label'] == 'automobile']
```

In [10]:
```
cat_data = image_train[image_train['label'] == 'cat']
```

In [11]:
```
bird_data = image_train[image_train['label'] == 'bird']
```

In [14]:
```
# let's review auto_data.

auto_data.show()
```

id	image	label	deep_features	image_array
dtype: int	dtype: Image	dtype: str	dtype: array	dtype: array
num_unique (est.): 505	First 4 images:	num_unique (est.): 1	num_unique (est.): 590,567	num_unique (est.): 255
num_undefined: 0		num_undefined: 0	num_undefined: 0	num_undefined: 0
min: 97		frequent items:	min: 0	min: 0
max: 49,919		automobile	max: 14.356	max: 255
median: 23,832			median: 0	median: 110
mean: 24,974.984			mean: 0.391	mean: 116.983
std: 14,433.575			std: 0.933	std: 69.71
distribution of values:			distribution of values (all sub-columns):	distribution of values (all sub-columns):



In [18]:
```
# now let's create deep learning model for each of the categories.

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

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

In [19]:
```
auto_model = graphlab.nearest_neighbors.create(auto_data,
                                               features=['deep_features'],
                                               label='id')
```

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

In [20]:
```
cat_model = graphlab.nearest_neighbors.create(cat_data,
                                              features=['deep_features'],
                                              label='id')
```

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

In [21]:
```
bird_model = graphlab.nearest_neighbors.create(bird_data,
                                               features=['deep_features'],
                                               label='id')
```

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

In [25]:
```
# let's review the dog data and find the nearest neighbours.  Here is the first image in the dog data.

dq = dog_data[0:1]
dq['image'].show()
```

All 1 images in <*SArray*>

In [ ]:

In [26]:
```
# now let's find out similar images using deep learning.  Remember, it doesn't have lable any more.
# In other words, it has same label, so not useful.

dog_model.query(dq)
```

```
Starting pairwise querying.

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

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

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

| 0            | 1       | 0.196464    | 38.725ms     |

| Done         |         | 100         | 131.166ms    |

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

Out[26]:

query_label	reference_label	distance	rank
0	70	0.0	1
0	19437	37.563553912	2
0	12088	41.9196580036	3
0	26747	42.3712394673	4
0	6184	42.8099391825	5

[5 rows x 4 columns]

[5 rows x 4 columns]

In [28]:
```
# let's create a function to make it easier to find neighbours and find out the quality of results.

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

In [30]:
```
show_neighbours(dog_model, dog_data,0)
```

```
Starting pairwise querying.

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

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

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

| 0            | 1       | 0.196464    | 18.768ms     |

| Done         |         | 100         | 105.547ms    |

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

All 5 images in *<SArray>*

In [31]:
```
show_neighbours(dog_model, dog_data,135)
```

```
Starting pairwise querying.

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

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

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

| 0            | 1       | 0.196464    | 13.184ms     |

| Done         |         | 100         | 98.114ms     |

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

All 5 images in *<SArray>*

In [ ]:
```
# wow, very good results!!!
```

In [ ]:
```
# now let's play with
```

In [32]:
```
show_neighbours(auto_model, auto_data,11)
```

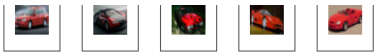
```
Starting pairwise querying.

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

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

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

| 0            | 1       | 0.196464    | 10.041ms     |

| Done         |         | 100         | 81.863ms     |

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

All 5 images in *<SArray>*

In [33]:
```
show_neighbours(auto_model, auto_data,428)
```

```
Starting pairwise querying.

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

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

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

| 0            | 1       | 0.196464    | 7.704ms      |

| Done         |         | 100         | 80.172ms     |

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

All 5 images in *<SArray>*



In [ ]:
```
# Nice!!
```

In [35]:
```
show_neighbours(cat_model, cat_data,211)
```

```
Starting pairwise querying.
+--------------+---------+-------------+--------------+
| Query points | # Pairs | % Complete. | Elapsed Time |
+--------------+---------+-------------+--------------+
| 0            | 1       | 0.196464    | 12.857ms     |
| Done         |         | 100         | 77.263ms     |
+--------------+---------+-------------+--------------+
```

All 5 images in <*SArray*>

In [ ]:
```
# very good.
```

In [37]:
```
show_neighbours(bird_model, bird_data,68)
```

```
Starting pairwise querying.
+--------------+---------+-------------+--------------+
| Query points | # Pairs | % Complete. | Elapsed Time |
+--------------+---------+-------------+--------------+
| 0            | 1       | 0.209205    | 21.044ms     |
| Done         |         | 100         | 101.137ms    |
+--------------+---------+-------------+--------------+
```

All 5 images in <*SArray*>

In [ ]:
```
# nice.
```

In [40]:
```
show_neighbours(bird_model, bird_data,200)
```

```
Starting pairwise querying.
+--------------+---------+-------------+--------------+
| Query points | # Pairs | % Complete. | Elapsed Time |
+--------------+---------+-------------+--------------+
| 0            | 1       | 0.209205    | 21.264ms     |
| Done         |         | 100         | 91.636ms     |
+--------------+---------+-------------+--------------+
```

All 5 images in <*SArray*>

In [ ]:
```
#cool!!  Love the results.
```

Now a big challenge. Measure the accuracy among these different models.

In [42]:
```
# first let's load the train data.
image_test = graphlab.SFrame('image_test_data/')
```

In [43]:
```
# we already created training data for each category.  now let's create test data for each category.
dog_test_data = image_test[image_test['label'] == 'dog']
```

In [44]:
```
auto_test_data = image_test[image_test['label'] == 'automobile']
```

In [45]:
```
cat_test_data = image_test[image_test['label'] == 'cat']
```

In [46]:
```
bird_test_data = image_test[image_test['label'] == 'bird']
```

In [47]:
```
# in above section, we have found nearest neighbour for one image at a time.
# in this section, we will call model to find all nearest neighbour for each image.
```

In [62]:
```
# First use dog model to find out nearest 1 neighbour.
# We will find the 1 (one) neighbour (k=1).
dog_dog_nbrs = dog_model.query(dog_test_data,k=1)
```

```
Starting blockwise querying.

max rows per data block: 4348

number of reference data blocks: 4

number of query data blocks: 1

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

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

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

| 1000         | 127000  | 24.9509     | 429.574ms    |

| Done         | 509000  | 100         | 513.197ms    |

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

In [61]:
```
# Use Cat Model on dog data.
# We will find the 1 (one) neighbour (k=1).
dog_cat_nbrs = cat_model.query(dog_test_data,k=1)
```

```
Starting blockwise querying.

max rows per data block: 4348

number of reference data blocks: 4

number of query data blocks: 1

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

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

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

| 1000         | 127000  | 24.9509     | 387.093ms    |

| Done         | 509000  | 100         | 430.458ms    |

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

In [57]:
```
# Use Auto Model on dog data.
dog_auto_nbrs = auto_model.query(dog_test_data,k=1)
```

```
Starting blockwise querying.

max rows per data block: 4348

number of reference data blocks: 4

number of query data blocks: 1

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

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

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

| 1000         | 127000  | 24.9509     | 375.754ms    |

| Done         | 509000  | 100         | 406.031ms    |

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

In [58]:
```
# Use Bird Model on dog data.
dog_bird_nbrs = bird_model.query(dog_test_data,k=1)
```

```
Starting blockwise querying.

max rows per data block: 4348

number of reference data blocks: 4

number of query data blocks: 1

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

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

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

| 1000         | 119000  | 24.8954     | 382.2ms      |

| Done         | 478000  | 100         | 419.964ms    |

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

In [59]:
```
dog_cat_nbrs.show()
```

query_label		reference_label		distance		rank	
dtype:	int	dtype:	int	dtype:	float	dtype:	int
num_unique (est.):	993	num_unique (est.):	265	num_unique (est.):	999	num_unique (est.):	1
num_undefined:	0	num_undefined:	0	num_undefined:	0	num_undefined:	0
min:	0	min:	33	min:	28.177	min:	1
max:	999	max:	49,840	max:	52.729	max:	1
median:	500	median:	26,336	median:	36.672	median:	1
						mean:	1

mean:	499.5
std:	288.675

distribution of values:

mean:	25,360.818
std:	15,317.822

distribution of values:

mean:	37.003
std:	3.721

distribution of values:

std:	0

distribution of values:

In [66]:
```
# now let's find out the distance for dog from dog data, cat data, auto data and bird data.

dog_distance = graphlab.SFrame({'dog_dog': dog_dog_nbrs['distance'],
                                'dog_cat': dog_cat_nbrs['distance'],
                                'dog_auto': dog_auto_nbrs['distance'],
                                'dog_bird': dog_bird_nbrs['distance'],
                               })
```

In [67]:
```
dog_distance.head()
```

Out[67]:

dog_auto	dog_bird	dog_cat	dog_dog
41.9579761457	41.7538647304	36.4196077068	33.4773590373
46.0021331807	41.3382958925	38.8353268874	32.8458495684
42.9462290692	38.6157590853	36.9763410854	35.0397073189
41.6866060048	37.0892269954	34.5750072914	33.9010327697
39.2269664935	38.272288694	34.778824791	37.4849250909
40.5845117698	39.1462089236	35.1171578292	34.945165344
45.1067352961	40.523040106	40.6095830913	39.0957278345
41.3221140974	38.1947918393	39.9036867306	37.7696131032
41.8244654995	40.1567131661	38.0674700168	35.1089144603
45.4976929401	45.5597962603	42.7258732951	43.2422832585

[10 rows x 4 columns]

In [78]:
```
# as you can see, dog_dog has the lowest distance which make sense.
# but not all rows.  The last row, dog_cat has lower distance than dog_dog.
# That means, dog image was closer to cat than dog!
# let's find out how many times dog_dog wins over other images.

def dog_wins(row):
    if (
        (row['dog_dog'] - row['dog_cat']) <= 0 and
        (row['dog_dog'] - row['dog_bird']) <= 0 and
        (row['dog_dog'] - row['dog_auto']) <= 0
    ):
        return 1
    else:
        return 0
```

In [81]:
```
dog_distance.apply(dog_wins).sum()
```

Out[81]:
```
678
```