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

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

Code CellToolbar

Deep learning using neural networks.

author: bhavesh patel

we will use images and its related tags to predict what the image is about.

we will use neural network for deep learning. Deep features is very interesting and see how it improves prediction.

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

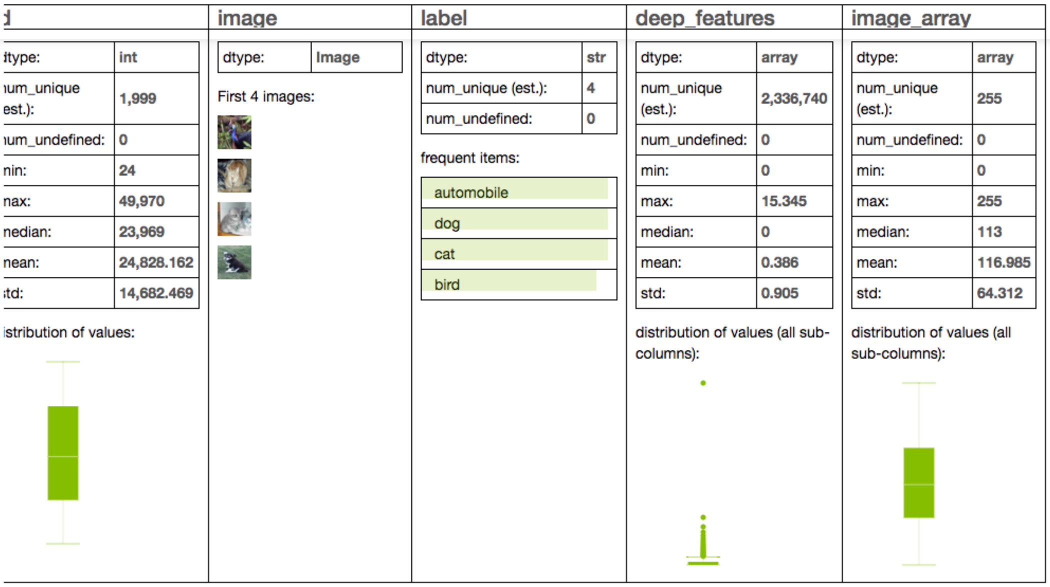
In [2]:
```
# Limit number of worker processes. This preserves system memory, which prevents hosted notebooks from crashing.
graphlab.set_runtime_config('GRAPHLAB_DEFAULT_NUM_PYLAMBDA_WORKERS', 4)
```

```
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_1481406886.log
```

In [5]:
```
# now let's load the images from CIFAR-10 dataset, but its reduced to four categories: cat, bird, automobile, dog.
# it is already split into training dataset and test dataset.

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

In [6]:
```
# set output local to here.
graphlab.canvas.set_target('ipynb')
```

In [8]:
```
# let's view the data.
image_train.show()
```



d

dtype:	int
um_unique est.):	1,999
um_undefined:	0
nin:	24
nax:	49,970
nedian:	23,969
nean:	24,828.162
std:	14,682.469

istribution 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 [9]:
```
# before we build model to predict, let's see what first three images are.
image_test[0:3]['image'].show()
```

All 3 images in <*SArray*>

In [10]:
```
# its cat, car, cat. noted.
# to confirm, here are the labels.
image_test[0:3]['label']
```

Out[10]:
```
dtype: str
Rows: 3
['cat', 'automobile', 'cat']
```

In [12]:
```
# now let's build ML model to train classifier.

image_classifier_model = graphlab.logistic_classifier.create(image_train, target='label',
                                                              features=['image_array'])
```

```
PROGRESS: Creating a validation set from 5 percent of training data. This may take a while.
```

```
            You can set ``validation_set=None`` to disable validation tracking.

WARNING: The number of feature dimensions in this problem is very large in comparison with the number of examples. Un
less an appropriate regularization value is set, this model may not provide accurate predictions for a validation/tes
t set.

Logistic regression:

--------------------------------------------------------

Number of examples          : 1910

Number of classes           : 4

Number of feature columns   : 1

Number of unpacked features : 3072

Number of coefficients    : 9219

Starting L-BFGS

--------------------------------------------------------

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

| Iteration | Passes   | Step size | Elapsed Time | Training-accuracy | Validation-accuracy |

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

| 1         | 6        | 0.000016  | 2.888895     | 0.331414          | 0.389474            |

| 2         | 8        | 1.000000  | 3.842132     | 0.383246          | 0.421053            |

| 3         | 9        | 1.000000  | 4.346528     | 0.408377          | 0.378947            |

| 4         | 10       | 1.000000  | 4.912695     | 0.445550          | 0.368421            |

| 5         | 11       | 1.000000  | 5.510324     | 0.447120          | 0.368421            |

| 6         | 12       | 1.000000  | 6.038188     | 0.465969          | 0.431579            |

| 10        | 16       | 1.000000  | 8.001951     | 0.521990          | 0.526316            |

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

TERMINATED: Iteration limit reached.

This model may not be optimal. To improve it, consider increasing `max_iterations`.
```

In [13]:

```
# now use the model to predict.

image_classifier_model.predict(image_test[0:3])
```

Out[13]:

```
dtype: str
Rows: 3
['bird', 'cat', 'bird']
```

In [14]:

```
# well, that's horrible accuracy.  :(
# let's evluate to find out.

image_classifier_model.evaluate(image_test)
```

Out[14]:

```
{'accuracy': 0.48075, 'auc': 0.7235272916666664, 'confusion_matrix': Columns:
        target_label    str
        predicted_label str
        count   int

 Rows: 16

 Data:
 +--------------+-----------------+-------+
 | target_label | predicted_label | count |
 +--------------+-----------------+-------+
 |     bird     |       dog       |  198  |
 |     dog      |       cat       |  239  |
 |     bird     |    automobile   |  112  |
 |  automobile  |    automobile   |  607  |
 |     cat      |       dog       |  303  |
 |     dog      |       dog       |  431  |
 |     dog      |    automobile   |   88  |
 |     bird     |       bird      |  529  |
 |  automobile  |       bird      |  118  |
 |     bird     |       cat       |  161  |
 +--------------+-----------------+-------+
 [16 rows x 3 columns]
 Note: Only the head of the SFrame is printed.
 You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns., 'f1_score': 0.4807160516374978,
 'log_loss': 1.2065411828057908, 'precision': 0.48193676238170613, 'recall': 0.48075, 'roc_curve': Columns:
        threshold       float
        fpr     float
        tpr     float
        p       int
        n       int
        class   int

 Rows: 400004

 Data:
 +-----------+-----+-----+------+------+-------+
 | threshold | fpr | tpr |  p   |  n   | class |
 +-----------+-----+-----+------+------+-------+
 |    0.0    | 1.0 | 1.0 | 1000 | 3000 |   0   |
 |   1e-05   | 1.0 | 1.0 | 1000 | 3000 |   0   |
 |   2e-05   | 1.0 | 1.0 | 1000 | 3000 |   0   |
 |   3e-05   | 1.0 | 1.0 | 1000 | 3000 |   0   |
 |   4e-05   | 1.0 | 1.0 | 1000 | 3000 |   0   |
 |   5e-05   | 1.0 | 1.0 | 1000 | 3000 |   0   |
 |   6e-05   | 1.0 | 1.0 | 1000 | 3000 |   0   |
 |   7e-05   | 1.0 | 1.0 | 1000 | 3000 |   0   |
 |   8e-05   | 1.0 | 1.0 | 1000 | 3000 |   0   |
 |   9e-05   | 1.0 | 1.0 | 1000 | 3000 |   0   |
+-----------+-----+-----+------+------+-------+
[400004 rows x 6 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.}
```

In [15]:
```
# ok so only 48% accuracy!  Not good.
```

In [16]:
```
# now let's use deep features.  Borrow it!
# first load the model.
deep_learning_model = graphlab.load_model('http://s3.amazonaws.com/GraphLab-Datasets/deeplearning/imagenet_model_iter45
```

```
Downloading http://s3.amazonaws.com/GraphLab-Datasets/deeplearning/imagenet_model_iter45/dir_archive.ini to /var/tmp/
graphlab-admin/6030/ee1a16f0-618c-48c7-a5ec-80a8f621dcbe.ini

Downloading http://s3.amazonaws.com/GraphLab-Datasets/deeplearning/imagenet_model_iter45/objects.bin to /var/tmp/grap
hlab-admin/6030/6e5de8fd-5303-4706-bddf-fc5cd6d192fc.bin
```

In [17]:
```
# now let's extract the features for our data based on this model.
image_train['bp_deep_features'] = deep_learning_model.extract_features(image_train)
```

```
Images being resized.
```

In [18]:
```
image_train.show()
```

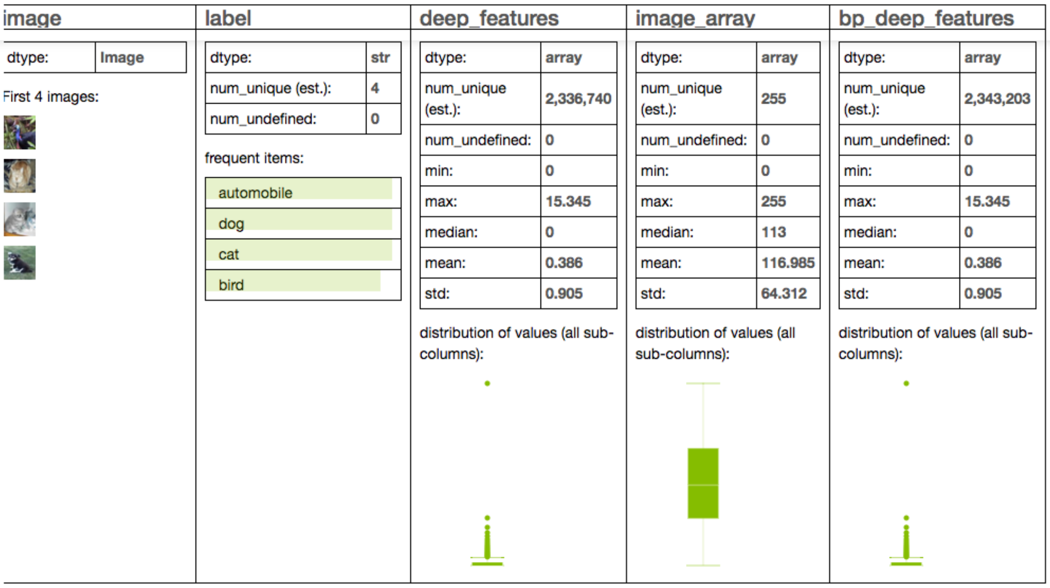


image	label	deep_features	image_array	bp_deep_features
dtype: Image	dtype: str	dtype: array	dtype: array	dtype: array
First 4 images:	num_unique (est.): 4	num_unique (est.): 2,336,740	num_unique (est.): 255	num_unique (est.): 2,343,203
	num_undefined: 0	num_undefined: 0	num_undefined: 0	num_undefined: 0
	frequent items:	min: 0	min: 0	min: 0
	automobile	max: 15.345	max: 255	max: 15.345
	dog	median: 0	median: 113	median: 0
	cat	mean: 0.386	mean: 116.985	mean: 0.386
	bird	std: 0.905	std: 64.312	std: 0.905
		distribution of values (all sub-columns):	distribution of values (all sub-columns):	distribution of values (all sub-columns):

In [20]:
```
# It took long time to process the new model.  But finally it did!
# Let's use this deep featuers, which are borrowed from other model.

deep_feature_model=graphlab.logistic_classifier.create(image_train,
                                                        features=['bp_deep_features'],
                                                        target='label')
```

```
PROGRESS: Creating a validation set from 5 percent of training data. This may take a while.
          You can set ``validation_set=None`` to disable validation tracking.

WARNING: The number of feature dimensions in this problem is very large in comparison with the number of examples. Un
less an appropriate regularization value is set, this model may not provide accurate predictions for a validation/tes
t set.

WARNING: Detected extremely low variance for feature(s) 'bp_deep_features' because all entries are nearly the same.
Proceeding with model training using all features. If the model does not provide results of adequate quality, exclude
 the above mentioned feature(s) from the input dataset.

Logistic regression:

--------------------------------------------------------

Number of examples          : 1918

Number of classes           : 4

Number of feature columns   : 1

Number of unpacked features : 4096

Number of coefficients    : 12291

Starting L-BFGS

--------------------------------------------------------

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

| Iteration | Passes   | Step size | Elapsed Time | Training-accuracy | Validation-accuracy |

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

| 1         | 5        | 0.000130  | 2.699571     | 0.757039          | 0.678161            |

| 2         | 9        | 0.250000  | 5.461219     | 0.765902          | 0.724138            |

| 3         | 10       | 0.250000  | 6.176370     | 0.773723          | 0.701149            |

| 4         | 11       | 0.250000  | 6.907332     | 0.778936          | 0.712644            |
| 5         | 12       | 0.250000  | 7.658626     | 0.789364          | 0.712644            |
| 6         | 13       | 0.250000  | 8.439804     | 0.800313          | 0.712644            |
| 7         | 14       | 0.250000  | 9.485471     | 0.819082          | 0.701149            |
| 8         | 15       | 0.250000  | 10.499534    | 0.842544          | 0.735632            |
| 9         | 16       | 0.250000  | 11.402847    | 0.873827          | 0.747126            |
| 10        | 17       | 0.250000  | 12.264933    | 0.895203          | 0.735632            |
+-----------+----------+-----------+--------------+-------------------+---------------------+
TERMINATED: Iteration limit reached.
This model may not be optimal. To improve it, consider increasing `max_iterations`.
```

In [21]:

```
deep_feature_model.predict(image_test[0:3])
```

Out[21]:

```
dtype: str
Rows: 3
['cat', 'cat', 'cat']
```

In [22]:

```
# ok good improvement.  Let's compare all three.
# real values are: cat, car, cat
# our training data predicted it as: bird, cat, bird
# our deep feature model predicted, cat, cat, cat -> not bad, but I was hoping better!
```

In [24]:

```
# now let's find accuracy of this model.

deep_feature_model.evaluate(image_test)
```

Out[24]:

```
{'accuracy': 0.25, 'auc': 0.5, 'confusion_matrix': Columns:
        target_label    str
        predicted_label str
        count   int

 Rows: 4

 Data:
 +--------------+-----------------+-------+
 | target_label | predicted_label | count |
 +--------------+-----------------+-------+
 |     dog      |       cat       |  1000 |
 |     bird     |       cat       |  1000 |
 |  automobile  |       cat       |  1000 |
 |     cat      |       cat       |  1000 |
 +--------------+-----------------+-------+
 [4 rows x 3 columns], 'f1_score': 0.1, 'log_loss': 1.466788500595805, 'precision': 0.25, 'recall': 0.25, 'roc_curv
e': Columns:
        threshold       float
        fpr     float
        tpr     float
        p       int
        n       int
        class   int

 Rows: 400004

 Data:
 +-----------+-----+-----+------+------+-------+
 | threshold | fpr | tpr |  p   |  n   | class |
 +-----------+-----+-----+------+------+-------+
 |    0.0    | 1.0 | 1.0 | 1000 | 3000 |   0   |
 |   1e-05   | 1.0 | 1.0 | 1000 | 3000 |   0   |
 |   2e-05   | 1.0 | 1.0 | 1000 | 3000 |   0   |
 |   3e-05   | 1.0 | 1.0 | 1000 | 3000 |   0   |
 |   4e-05   | 1.0 | 1.0 | 1000 | 3000 |   0   |
 |   5e-05   | 1.0 | 1.0 | 1000 | 3000 |   0   |
 |   6e-05   | 1.0 | 1.0 | 1000 | 3000 |   0   |
 |   7e-05   | 1.0 | 1.0 | 1000 | 3000 |   0   |
 |   8e-05   | 1.0 | 1.0 | 1000 | 3000 |   0   |
 |   9e-05   | 1.0 | 1.0 | 1000 | 3000 |   0   |
 +-----------+-----+-----+------+------+-------+
 [400004 rows x 6 columns]
 Note: Only the head of the SFrame is printed.
 You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.}
```

In [25]:

```
# only 25%.  That's not good.  That's because my computer didn't continue to iterate.
```

In [ ]:

```
# let's use the deep features, which were part of the model and see what we get.
```

In [31]:

```
deep_feature_precalculated_model = graphlab.logistic_classifier.create(image_train,
                                                              features =['deep_features'],
                                                              target='label')
```

```
PROGRESS: Creating a validation set from 5 percent of training data. This may take a while.
          You can set ``validation_set=None`` to disable validation tracking.

WARNING: The number of feature dimensions in this problem is very large in comparison with the number of examples. Un
less an appropriate regularization value is set, this model may not provide accurate predictions for a validation/tes
t set.

WARNING: Detected extremely low variance for feature(s) 'deep_features' because all entries are nearly the same.
Proceeding with model training using all features. If the model does not provide results of adequate quality, exclude
 the above mentioned feature(s) from the input dataset.

Logistic regression:

--------------------------------------------------------

Number of examples          : 1889

Number of classes           : 4
Number of feature columns   : 1

Number of unpacked features : 4096

Number of coefficients    : 12291

Starting L-BFGS

--------------------------------------------------------

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

| Iteration | Passes   | Step size | Elapsed Time | Training-accuracy | Validation-accuracy |

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

| 1         | 5        | 0.000132  | 2.434330     | 0.741133          | 0.715517            |

| 2         | 9        | 0.250000  | 5.056429     | 0.772896          | 0.775862            |

| 3         | 10       | 0.250000  | 5.836083     | 0.775543          | 0.775862            |

| 4         | 11       | 0.250000  | 6.608528     | 0.779778          | 0.767241            |

| 5         | 12       | 0.250000  | 7.481547     | 0.790895          | 0.775862            |

| 6         | 13       | 0.250000  | 8.274079     | 0.801482          | 0.784483            |

| 7         | 14       | 0.250000  | 8.980571     | 0.824246          | 0.758621            |

| 8         | 15       | 0.250000  | 9.986029     | 0.838539          | 0.793103            |

| 9         | 16       | 0.250000  | 10.892873    | 0.852832          | 0.793103            |

| 10        | 17       | 0.250000  | 11.719468    | 0.872949          | 0.793103            |

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

TERMINATED: Iteration limit reached.

This model may not be optimal. To improve it, consider increasing `max_iterations`.
```

In [32]:

```
deep_feature_precalculated_model.predict(image_test[0:3])
```

Out[32]:

```
dtype: str
Rows: 3
['cat', 'automobile', 'cat']
```

In [ ]:

```
# wow-> finally ML got it.
```

In [33]:

```
# let's see accuracy of this model.
```

In [35]:

```
deep_feature_precalculated_model.evaluate(image_test)
```

Out[35]:

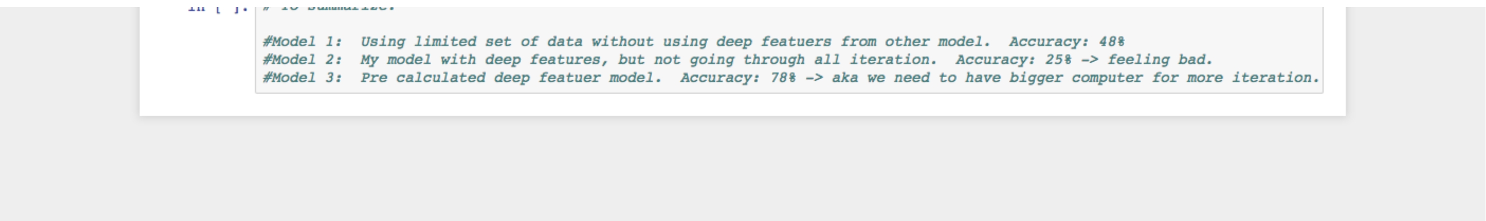
```
{'accuracy': 0.784, 'auc': 0.9384483749999979, 'confusion_matrix': Columns:
        target_label    str
        predicted_label str
        count   int

 Rows: 16

 Data:
 +--------------+-----------------+-------+
 | target_label | predicted_label | count |
 +--------------+-----------------+-------+
 |  automobile  |       cat       |   14  |
 |     bird     |       dog       |   58  |
 |     cat      |       bird      |   69  |
 |  automobile  |       dog       |   7   |
 |     cat      |    automobile   |   33  |
 |     dog      |       bird      |   44  |
 |     bird     |       cat       |  130  |
 |     dog      |    automobile   |   20  |
 |     dog      |       dog       |  716  |
 |     cat      |       dog       |  222  |
 +--------------+-----------------+-------+
 [16 rows x 3 columns]
 Note: Only the head of the SFrame is printed.
 You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns., 'f1_score': 0.7841899124404468,
 'log_loss': 0.6112328671477912, 'precision': 0.78548595574037, 'recall': 0.784, 'roc_curve': Columns:
        threshold       float
        fpr     float
        tpr     float
        p       int
        n       int
        class   int

 Rows: 400004

 Data:
 +-----------+----------------+-----+------+------+-------+
 | threshold |      fpr       | tpr |  p   |  n   | class |
 +-----------+----------------+-----+------+------+-------+
 |    0.0    |      1.0       | 1.0 | 1000 | 3000 |   0   |
 |   1e-05   | 0.966333333333 | 1.0 | 1000 | 3000 |   0   |
 |   2e-05   | 0.954333333333 | 1.0 | 1000 | 3000 |   0   |
 |   3e-05   |     0.946      | 1.0 | 1000 | 3000 |   0   |
 |   4e-05   |      0.94      | 1.0 | 1000 | 3000 |   0   |
 |   5e-05   | 0.931666666667 | 1.0 | 1000 | 3000 |   0   |
 |   6e-05   | 0.928333333333 | 1.0 | 1000 | 3000 |   0   |
 |   7e-05   | 0.925333333333 | 1.0 | 1000 | 3000 |   0   |
 |   8e-05   | 0.919666666667 | 1.0 | 1000 | 3000 |   0   |
 |   9e-05   | 0.918666666667 | 1.0 | 1000 | 3000 |   0   |
 +-----------+----------------+-----+------+------+-------+
 [400004 rows x 6 columns]
 Note: Only the head of the SFrame is printed.
 You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.}
```

In [36]:

```
# wow 78% accuracy.  That's very good.
```

In [ ]:

```
# To summarize:
```



```
In [ ]: # To summarize:

        #Model 1:  Using limited set of data without using deep featuers from other model.  Accuracy: 48%
        #Model 2:  My model with deep features, but not going through all iteration.  Accuracy: 25% -> feeling bad.
        #Model 3:  Pre calculated deep featuer model.  Accuracy: 78% -> aka we need to have bigger computer for more iteration.
```