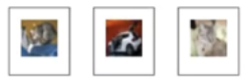
| Deep Learning for Image Classification | |
|---|---|
| Load the data | 1. Load image_train_data and image_test_data<br>2. *.show()* to see the images |
| Training the Data, initial pass at prediction | 1. Train the data<br>`raw_pixel_model = graphlab.logistic_classifier.create(image_train,target='`<br>`features=['image_array'])`<br>`PROGRESS: Creating a validation set from 5 percent of training data. This`<br>`ake a while.`<br>`You can set ``validation_set=None`` to disable validation tracki`<br>2. Take some images from the test data set and see what the classifier says they are.<br>`image_test[0:3]['image'].show()`<br>All 3 images in *<temporary SArray>*<br>  <br>`In [8]: image_test[0:3]['label']`<br>`Out[8]: dtype: str`<br>`Rows: 3`<br>`['cat', 'automobile', 'cat']`<br>3. Let's see what the model predicts for the three images.<br>`In [9]: raw_pixel_model.predict(image_test[0:3])`<br>`Out[9]: dtype: str`<br>`Rows: 3`<br>`['bird', 'cat', 'bird']`<br>Very off…<br>4. Evaluate the raw pixel model on test data, very low accuracy<br>`In [10]: raw_pixel_model.evaluate(image_test)`<br>`Out[10]: {'accuracy': 0.468, 'confusion_matrix': Columns:`<br>`target_label int`<br>`predicted_label int`<br>`count int`<br>`Rows: 16` |
| Improve The Model using Deep Features | 1. Pull the images to train<br>`deep_learning_model = graphlab.load_model('imagenet_model')`<br>`image_train['deep_features'] = deep_learning_model.extract_features(image_train)`<br>`image_train.head()`<br>id: 24 \| image: Height: 32 Width: 32 \| label: bird \| deep_features: [0.242871761322, 1.09545373917, 0.0, ...] \| image_array: [73.0, 77.0, 58.0, 71.0, 68.0, 50.0, 77.0, 69.0, ...]<br>2. Code to train the model |

<table>
<tr><td></td><td>

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

3. Apply the deep features model to the first few images of test set and see if it's more accurate now

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

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

4. Show the accuracy level

```
deep_features_model.evaluate(image_test)
```

```
{'accuracy': 0.7845, 'confusion_matrix': Columns:
        target_label    int
        predicted_label int
        count   int
```

</td></tr>
<tr><td colspan="2">Deep Features for Image Retrieval</td></tr>
<tr><td>Load the Data</td><td>

1. Load image_train_data and image_test_data
2. *.show()* to see the images

</td></tr>
<tr><td>Creating a nearest neighbors model</td><td>

1. Train a nearest neighbor model (knn_model) for retrieving images using deep features

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

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

2. Retrieve an image to find the nearest images on

```
graphlab.canvas.set_target('ipynb')
cat = image_train[18:19]
cat['image'].show()
```

3. Find the nearest images for the cat

```
knn_model.query(cat)
```

| query_label | reference_label | distance | rank |
|---|---|---|---|
| 0 | 384 | 0.0 | 1 |
| 0 | 6910 | 36.9403137951 | 2 |
| 0 | 39777 | 38.4634888975 | 3 |
| 0 | 36870 | 39.7559623119 | 4 |
| 0 | 41734 | 39.7866014148 | 5 |

4. Create a function that takes those reference labels and returns the image

```
def get_images_from_ids(query_result):
    return image_train.filter_by(query_result['reference_label'],'id')
```

```
cat_neighbors = get_images_from_ids(knn_model.query(cat))
```

```
PROGRESS: Starting pairwise querying...
PROGRESS: +--------------+---------+-------------+--------------+
PROGRESS: | Query points | # Pairs | % Complete. | Elapsed Time |
PROGRESS: +--------------+---------+-------------+--------------+
PROGRESS: | 0            | 1       | 0.0498753   | 7.302ms      |
PROGRESS: | Done         |         | 100         | 69.934ms     |
PROGRESS: +--------------+---------+-------------+--------------+
```

```
cat_neighbors['image'].show()
```

5. Next time we look for an image, it will return the pictures

</td></tr>
</table>

<table>
<tr>
<td></td>
<td>

```
car = image_train[8:9]
car['image'].show()
```

All 1 images in <*temporary SArray*>

```
get_images_from_ids(knn_model.query(car))['image'].show()
```

All 5 images in <*temporary SArray*>


</td>
</tr>
<tr>
<td>Python Lambda for Image Retrieval</td>
<td>

Function:

```
In [14]: lambda i: get_images_from_ids(knn_model.query(image_train[i:i+1]))['image'].show()
```

Find the car (image #8)

```
show_neighbors = lambda i: get_images_from_ids(knn_model.query(image_train[i:i+1]
```

```
show_neighbors(8)
```

All 5 images in <*temporary SArray*>


</td>
</tr>
</table>