Deep features for image retrieval

In this module, we focused on using deep learning to create non-linear features to improve the performance of machine learning. We also saw how transfer learning techniques can be applied to use deep features learned with one dataset to get great performance on a different dataset. We also built an iPython notebooks for both image retrieval and image classification tasks on real datasets.

In this assignment, we are going to build new image retrieval models and explore their results on different parts of our image dataset. These techniques will be used at the core of the intelligent application in your capstone project.

Follow the rest of the instructions on this page to complete your program. When you are done, instead of uploading your code, you will answer a series of quiz questions (see the quiz after this reading) to document your completion of this assignment. The instructions will indicate what data to collect for answering the quiz.

Learning outcomes

- Execute image retrieval code with the iPython notebook
- Use the .sketch_summary() method to view statistics of data
- Load and transform real, image data
- Build image retrieval models using nearest neighbor search and deep features
- Compare the results of various image retrieval models
- Use the .apply() and .sum() methods on SFrames to compute functions of the data.

Resources you will need

You will need to install the software tools or use the free Amazon EC2 machine. Instructions for both options are provided in the reading for Module 1.

Download the data and starter code

Before getting started, you will need to download the dataset and the starter iPython notebook that we used in the module.

- Download the wikipedia dataset with training images here in SFrame format: **image_train_data.zip**
- Download the wikipedia dataset with test images herein SFrame format: **image_test_data.zip**
- Download the image retrieval notebook from the module here: Deep Features for Image Classification.ipynb
- Download the image retrieval notebook from the module here: Deep Features for Image Retrieval.ipynb
- Save all of these files in the same directory (where you are calling iPython notebook from) and unzip the data files. **Not sure where to save the files? See this guide.**

Now you are ready to get started!

Note: If you would rather use other ML tools...

You are welcome to use any ML tool for this course, such as scikit-learn. Though, as discussed in the intro module, *we strongly recommend you use iPython Notebook and GraphLab Create. (GraphLab Create is free for academic purposes.)*

If you are choosing to use other packages, we still recommend you use SFrame, which will allow you to scale to much larger datasets than Pandas. (Though, it's possible to use Pandas in this course, if your machine has sufficient memory.) The SFrame package is available in open-source under a permissive BSD license. So, you will *always* be able to use SFrames for free.

If you are not using SFrame, here is the dataset for this assignment in CSV format, so you can use Pandas or other options out there: image_train_data.csv and image_test_data.csv.

Watch the videos and explore the iPython notebooks on using deep features for image classification and retrieval

If you haven't done so yet, before you start, we recommend you watch the video where we go over the iPython notebooks from this module. You can then open up the iPython notebook we used and familiarize yourself with the steps we covered in these examples.

What you will do

Now you are ready! We are going do four tasks in this assignment. There are several results you need to gather along the way to enter into the quiz after this reading.

1. **Computing summary statistics of the data:** Sketch summaries are techniques for computing summary statistics of data very quickly. In GraphLab Create, SFrames and SArrays include a method:

```
.sketch_summary()
```

which computes such summary statistics. Using the training data, compute the sketch summary of the 'label' column and interpret the results. What's the least common category in the training data? ***Save this result to answer the quiz at the end.***

2. **Creating category-specific image retrieval models:** In most retrieval tasks, the data we have is unlabeled, thus we call these unsupervised learning problems. However, we have labels in this image dataset, and will use these to create one model for each of the 4 image categories, {dog','cat','automobile',bird'}. To start, follow these steps:

- Split the SFrame with the training data into 4 different SFrames. Each of these will contain data for 1 of the 4 categories above. *Hint: if you use a logical filter to select the rows where the 'label' column equals 'dog', you can create an SFrame with only the data for images labeled 'dog'.*
- Similarly to the image retrieval notebook you downloaded, you are going to create a nearest neighbor model using the 'deep_features' as the features, but this time create one such model for each category, using the corresponding subset of the training_data. *You can call the model with the 'dog' data the dog_model, the one with the 'cat' data the cat_model, as so on.*

You now have a nearest neighbors model that can find the nearest 'dog' to any image you give it, the dog_model; one that can find the nearest 'cat', the cat_model; and so on.

Using these models, answer the following questions. The cat image below is the first in the *test data*:

You can access this image, similarly to what we did in the iPython notebooks above, with this command:

```
image_test[0:1]
```

- What is the nearest 'cat' labeled image in the training data to the cat image above (the first image in the test data)? ***Save this result.***

Hint: When you query your nearest neighbors model, it will return a SFrame that looks something like this:

query_label	reference_label	distance	rank
0	34	42.9886641167	1
0	45	43.8444904098	2
0	251	44.2634660468	3
0	141	44.3777119559	4

To understand each column in this table, see this documentation. For this question, the 'reference_label' column will be important, since it provides the index of the nearest neighbors in the dataset used to train it. (In this case, the subset of the training data labeled 'cat'.)

- What is the nearest 'dog' labeled image in the training data to the cat image above (the first image in the test data)? ***Save this result.***

3. A simple example of nearest-neighbors classification: When we queried a nearest neighbors model, the 'distance' column in the table above shows the computed distance between the input and each of the retrieved neighbors. In this question, you will use these distances to perform a classification task, using the idea of a nearest-neighbors classifier.

- For the first image in the test data (image_test[0:1]), which we used above, compute the mean distance between this image at its 5 nearest neighbors that were labeled **'cat'** in the training data (similarly to what you did in the previous question). ***Save this result.***
- Similarly, for the first image in the test data (image_test[0:1]), which we used above, compute the mean distance between this image at its 5 nearest neighbors that were labeled **'dog'** in the training data (similarly to what you did in the previous question). ***Save this result.***
- On average, is the first image in the test data closer to its 5 nearest neighbors in the 'cat' data or in the 'dog' data? (In a later course, we will see that this is an example of what is called a k-nearest neighbors classifier, where we use the label of neighboring points to predict the label of a test point.)

4. [Challenging Question] Computing nearest neighbors accuracy using SFrame operations: A nearest neighbor classifier predicts the label of a point as the most common label of its nearest neighbors. In this question, we will measure the accuracy of a 1-nearest-neighbor classifier, i.e., predict the output as the label of the nearest neighbor in the training data. Although there are simpler ways of computing this result, we will go step-by-step here to introduce you to more concepts in nearest neighbors and SFrames, which will be useful later in this Specialization.

- **Training models:** For this question, you will need the nearest neighbors models you learned above on the training data, i.e., the dog_model, cat_model, automobile_model and bird_model.
- **Spliting test data by label:** Above, you split the train data SFrame into one SFrame for images labeled 'dog', another for those labeled 'cat', etc. Now, do the same for the test data. You can call the resulting SFrames

```
image_test_cat, image_test_dog, image_test_bird, image_test_automobile
```

- **Finding nearest neighbors in the training set for each part of the test set:** Thus far, we have queried, e.g.,

```
dog_model.query()
```

our nearest neighbors models with a single image as the input, but you can actually query with a whole set of data, and it will find the nearest neighbors for each data point. Note that the input index will be stored in the 'query_label' column of the output SFrame.

Using this knowledge find the closest neighbor in to the dog test data using each of the trained models, e.g.,

```
dog_cat_neighbors = cat_model.query(image_test_dog, k=1)
```

finds 1 neighbor (that's what k=1 does) to the dog test images (*image_test_dog*) in the cat portion of the training data (used to train the *cat_model*).

Now, do this for every combination of the labels in the training and test data.

- **Create an SFrame with the distances from 'dog' test examples to the respective nearest neighbors in each class in the training data:** The 'distance' column in *dog_cat_neighbors* above contains the distance between each 'dog' image in the test set and its nearest 'cat' image in the *training set*. The question we want to answer is how many of the test set 'dog' images are closer to a 'dog' in the training set than to a 'cat', 'automobile' or 'bird'. So, next we will create an SFrame containing just these distances per data point. The goal is to create an SFrame called *dog_distances* with 4 columns:

i. *dog_distances['dog-dog']* ---- storing *dog_dog_neighbors['distance']*

ii. *dog_distances['dog-cat']* ---- storing *dog_cat_neighbors['distance']*

iii. *dog_distances['dog-automobile']* ---- storing *dog_automobile_neighbors['distance']*

iv. *dog_distances['dog-bird']* ---- storing *dog_bird_neighbors['distance']*

Hint: You can create a new SFrame from the columns of other SFrames by creating a dictionary with the new columns, as shown in this example:

```
new_sframe = graphlab.SFrame({'foo': other_sframe['foo'],'bar': some_other_sframe['bar']})
```

The resulting SFrame will look something like this:

dog-automobile	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

- **Computing the number of correct predictions using 1-nearest neighbors for the dog class:** Now that you have created the SFrame *dog_distances*, you will learn to use the method

```
.apply()
```

on this SFrame to iterate line by line and compute the number of 'dog' test examples where the distance to the nearest 'dog' was lower than that to the other classes. You will do this in three steps:

i. Consider one row of the SFrame dog_distances. Let's call this variable row. You can access each distance by calling, for example,

```
row['dog-cat']
```

which, in example table above, will have value equal to *36.4196077068* for the first row.

Create a function starting with

```
def is_dog_correct(row):
```

which returns 1 if the value for *row['dog-dog']* is lower than that of the other columns, and 0 otherwise. That is, returns 1 if this row is correctly classified by 1-nearest neighbors, and 0 otherwise.

ii. Using the function *is_dog_correct(row)*, you can check if 1 row is correctly classified. Now, you want to count how many rows are correctly classified. You could do a for loop iterating through each row and applying the function *is_dog_correct(row)*. This method will be really slow, because the SFrame is not optimized for this type of operation.

Instead, we will use the *.apply()* method to iterate the function is_dog_correct for each row of the SFrame. Read about using the *.apply()* method here.

iii. **Computing the number of correct predictions for 'dog':** You can now call:

```
dog_distances.apply(is_dog_correct)
```

which will return an SArray (a column of data) with a 1 for every correct row and a 0 for every incorrect one. You can call:

```
.sum()
```

on the result to get the total number of correctly classified 'dog' images in the test set!

Hint: To make sure your code is working correctly, if you were to do the two steps above in this question to count the number of correctly classified 'cat' images in the test data, instead of 'dog', the result would be 548.

- **Accuracy of predicting dog in the test data:** Using the work you did in this question, what is the accuracy of the 1-nearest neighbor classifier at classifying 'dog' images from the test set? ***Save this result to answer the quiz at the end.***