PROJECT REPORT

On

Content Based Image Retrieval

Presented by Mrinal Sourav for Pattern Recognition and Computer Vision Course.

Introduction

Content based Image Retrieval is the paradigm of image based search in which the contents of the image is given a higher priority instead of the tags associated with the image, if any. Although traditional approaches to CBIR, like histogram based searches could perform well with small datasets, with the advent of recent developments in CNNs that can extract features from images, it became viable to implement large scale Image based search engines.

The premise of the CBIR I have implemented is to take a query image from the user and retrieve the exact same image as well as other images with similar features from a relatively large set of images in a file system. For this project, I have used ResNet50, which is a pretrained CNN available in Keras. Using this CNN, I extracted the features and indexed a combined dataset of 23925 images. I then trained a neural network which can compute a similarity measure by regression on the difference of two image feature vectors. The results of the search engine are retrieved by sorting based on the similarity score and extracting a “topn” number of images from the sorted results.

The project could be broadly sectioned into phases of Data Collection, Indexing the Data, Preparation of the training data, Training the Neural Net and Executing Tests. Each of these phases are detailed below.

Tools Used

I used a Google Compute engine with Ubuntu to setup my python3 environment with the following packages:

1. Keras – to import and build Neural Network models
2. Tensorflow – the backend for the Keras library
3. Numpy – for array and image manipulations
4. Pickle – to store and retrieve the list of indexes
5. Matplotlib – to display images
6. Jupyter – the IDE for python.
7. Glob – to read folders of images
8. Re – to match regular expressions to parse folder structures and identify the hierarchy of image similarity.

Phase I - Dataset Collection

The dataset of images used to train the custom built Neural Network was borrowed from multiple datasets from the internet. This was done to ensure a variety of features that could be presented to the Neural Net to generalize on the differences between various images. Although a balanced representation of data is preferred while training a Neural net, the dataset I have used contains about 17000 samples of Dogs and 7000 samples of buildings. Following are the details of the dataset:

1. Subsection of ImageNet – The Dogs folder with 120 subfolders for different breeds of dogs.
2. Parts of San Francisco Landscape Dataset – I manually curated a set of 62 folders containing images of different buildings with an average of about 50 images per folder.
3. Parts of Oxford Building Dataset – I manually curated subset of 13 folders, containing various buildings and other artefacts from the "oxbuild" dataset.
4. A small part of the 'cdiscount' dataset made available for a Kaggle competition.

Phase II – Indexing the Data

The Jupyter Notebook code named "Folder_index" does the task of indexing the images.

- It starts with importing the pretrained ResNet50 model **without the top layers** from Keras.
- Helper functions are defined to index individual images and glob folders with hierarchical subfolders.
- Entire folders like 'Dogs', 'san_fran_selected' etc. are indexed into a list of tuples that look like:

```
[('cdiscount/1000010667/8.jpg',
  array([ 0.06156372,  0.41374597,  0.        , ...,  0.36921111,
          0.18318413,  0.13159379], dtype=float32)),
 ('cdiscount/1000010667/64.jpg',
  array([ 0.        ,  0.        ,  0.12313852, ...,  0.02729335,
          1.46558285,  0.19516996], dtype=float32)),
 ('cdiscount/1000010667/98.jpg',
  array([ 0.        ,  0.03851786,  0.        , ...,  0.        ,
          0.83109021,  0.3998245 ], dtype=float32)),
 ('cdiscount/1000010667/77.jpg',
  array([ 0.75042909,  0.27932882,  0.00150269, ...,  0.00286587,
          1.24573493,  0.01464425], dtype=float32)),
```

The first part of the tuple is the path of the image and the second part is the flattened array that is returned by ResNet on extracting the features of the image. This flattened array has 2048 values in it.

- The index list is finally dumped as a "search_index.pickle" file. [I later realized using tuples to store indexes meant that the indexes could not be stored as Numpy – this has repercussions with the runtime of the model itself and the Euclidean search used to evaluate the results as the indexes would now have to be retrieved one at a time.]

Phase III – Preparation of the Training Data

The code named "dataset_generator" deals with generating batches of training datasets from an N^2 comparisons of indexes in the indexes list. The **indexes need to be shuffled** before any operation because I created multiple training batches out of them. The same parsing of folders for hierarchy is done to assign different **synthetic** Ground Truths for each comparison of indexes.

Here's an explanation of the ground truths selected:

- For images that are exactly same (i.e. when they have the same name and come from the same subfolder in the same dataset), the train input contains an array of 0s and the train target is appended by the prime number '7'. The choice of prime numbers is arbitrary but the idea is to ensure a clean non-linear separation between different cases of similarity and the distance between images according to the folders they reside in.

- For images that come from the same subfolder in the 'san_fran' dataset the number '11' is assigned as the ground truth. I consider the images from the "san_fran" dataset as very clean definition of similar images of buildings. The original dataset contained images taken from a mobile vehicle that took continuous images from the streets of San Francisco. I then curated specific different buildings manually.

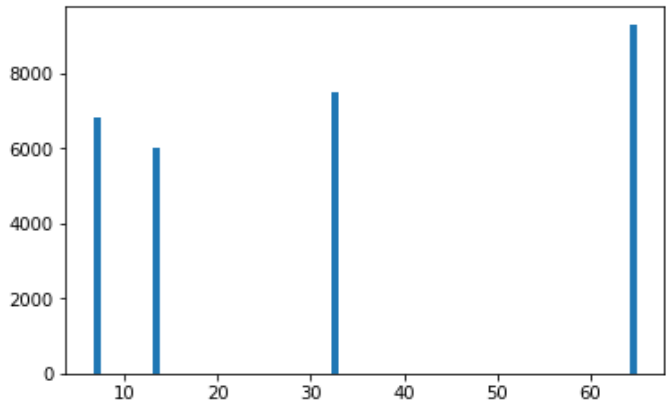
- For other cases where images come from the same subfolder but having different image names, I have assigned '17' as the ground truth.

- For images coming together from a different sub folder but from the same dataset I have assigned '37' as the differentiating value. Also, a random dropout is added to take only 1 in 95 comparisons. This is done to ensure that the training set remains relatively balanced while at the same time avoiding the issue of memory overload because I am doing an N^2 operation that takes a lot of resources. (There are way more out of subfolder comparisons than there are exact and similar matches)

- Finally, for images that come from completely different dataset a ground truth of "71" is assigned. **Notice that the differences in the ground truths is almost exponential**.

  The N^2 comparisons are done 1000 X 1000 comparisons at a time to avoid memory overload to create training batches with a distribution of ground truths that look like:

```
num, bins = np.histogram(train_target)
plt.bar(bins[:-1], num)
plt.show()
```



(This is just one training sample out of 23 created from 23X1000X1000 comparisons.)

Training the Neural Net

The neural network architecture in its final version is as below:

```
model = Sequential()
```

```
model.add(Dense(2048, input_dim = 2048, activation = 'sigmoid'))
model.add(Dense(1200, activation = 'sigmoid'))
model.add(Dropout(0.2))
model.add(Dense(700, activation = 'relu'))
model.add(Dropout(0.1))
model.add(Dense(300, activation = 'relu'))
model.add(Dense(30, activation = 'relu'))
model.add(Dense(1))
```

This a pyramid architecture that steadily decreases towards the output. The first two layers are sigmoid. The intuition behind choosing is that the input values of features are in “float” and all of them would be positive as the network would be input with the squared difference between the features of two images and as such I wanted to use some flexibility that a sigmoid function provides while keeping outputs positive. Dropouts have been added to avoid overfitting as over multiple tries during training I found that the Network overfits drastically with the dataset.
Relu layers are added to support a graceful regression towards the final output which is just one unit without any activation.

Once the training dataset was created, multiple tries of training were carried out. Initially while providing 10 epochs per batch in a loop I could see against the validation loss that my model was overfitting. Also, during training in batches, whenever there was overfitting to a degree, even though the current batch could train down to a loss of around 3, the next batch would diverge to a loss of 285 and find a “local minima”

there. I therefore had to take the extreme measure of only training with for 2 epochs per batch. Also, a file based version management of the model had to be carried out to update with each loop of the training dataset.

After training, the initial results seemed promising but then I compared against Euclidean distance method between images to see how my trained model was doing against that. I found that the Euclidean distance method was doing much better. However, after a few updates and tweaks the Neural Network started showing some good results when compared with test images against the Euclidean distance method.

The tests have been captured in the test and evaluation phase.

Evaluation and Tests

The “execute test” Jupyter notebook contains all the tests carried out on the ~23000 images that were indexed. Some test images (like the space needle) were also freshly indexed to check against samples that have 0 probability of being used for training. Initial few images are to test against the images that **may** have been encountered by the network during training. The uncertainty of which images were encountered comes from the fact that I have randomly dropped a lot of image feature comparisons to keep the training dataset balanced.

Most of the tests carried out were for top10 images retrieved.

I found that for some pictures the L2 Distance based search was performing better. However, for architectural images, the Neural Network retrieves more images from the same subclass. Even with the test images, the Neural Network can be seen to outperform L2 search.

Conclusions

Based on the limited set of tests performed one can observe the potential of Neural Networks that generalize to the test environment of the training. Much care needs to be taken to not overfit the Network but if trained with proper dataset and careful observation of parameters, I could see a drastic difference in the performance of the Neural Network. As the model is trained with very few sets of images (Dogs and buildings mostly), it does not perform well for images from other datasets. Increasing the variance in the dataset is the one scope I see that can further improve the Neural Network for a real-world implementation.

Also, further improvements in the CNNs ability to extract features better could also increase the performance of the neural network.

References

1. IEEE Paper on CBIR with CNN and SVM - http://ieeexplore.ieee.org/document/7924779/
2. San Francisco Landmark Dataset from Stanford - https://exhibits.stanford.edu/data/catalog/vn158kj2087
3. Oxford Building Dataset - http://www.robots.ox.ac.uk/~vgg/data/oxbuildings/
4. The “Dogs” sysnet from ImageNet - http://vision.stanford.edu/aditya86/ImageNetDogs/
5. ResNet50 CNN model on Keras - https://keras.io/applications/#resnet50