Project:

Finding Similar Images using Locality Sensitive Hashing Technique

Prakhar Dogra

Email: pdogra@gmu.edu

G Number: G01009586

Introduction

Main goal of the project is to apply a step by step approach of Shingling followed by Min-Hashing followed by Locality Sensitive Hashing to find similar images in a large database of unlabeled images.

Data Set Used

The datasets used for this project have been obtained from Kaggle's ImageNet competitions. Within each dataset there are three folders of images i.e., *train*, *test* and *val*. Exactly 14,000 images have been sampled from these folders (not all images from the same folder).

Each image is in JPEG format with most images ranging in width from 300 pixels to 800 pixels and in height from 250 pixels to 1200 pixels.

All the images are unique (no duplicates) and belong to a variety of different types of images.

File Weblinks:

1. https://www.kaggle.com/c/imagenet-object-localization-challenge/download/imagenet_object_localization.tar.gz
2. https://www.kaggle.com/c/imagenet-object-detection-challenge/download/imagenet_object_detection_train.tar.gz

Download the zip file. Extract it and adjust the path in the source file.

Method and Approach

Initially images were loaded into memory as numpy arrays along with their respective filenames.

Then the following two step by step approaches were implemented:

1. Shingling – Min Hashing - Local Sensitive Hashing (LSH)
2. Shingling – Random Hyperplanes - Local Sensitive Hashing (LSH)

A brief about these methods:

1. Shingling: Convert Images to sets
2. Min-Hashing: Convert large sets to smaller signatures
3. Random Hyper planes: Convert large sets to smaller sketches
4. Local Sensitive Hashing (LSH): focus on pairs of signatures that are likely to be very similar to avoid performance hit.

For the final step, two similarity measures were used to find similar images:

1. Jaccard Similarity Score (when using Min Hashing): It measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets
2. Cosine Similarity (when using Random Hyper Planes): It measures the similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them

All these methods will be explained in detail below.

1. SHINGLING

Given a positive integer *k* and a sequence of terms in a document *d*, define the *k*-shingles of *d* to be the set of all consecutive sequences of *k* terms in *d*. But it must be realized that shingling in documents is different from shingling in images.

So, shingling in images is defined as a set of all consecutive windows of size k X k pixels in image *i.* Therefore, an image of dimensions m X n will have $(m - k + 1)$ X $(n - k + 1)$ windows of size k X k pixels. Following is a good example to show shingling in images:

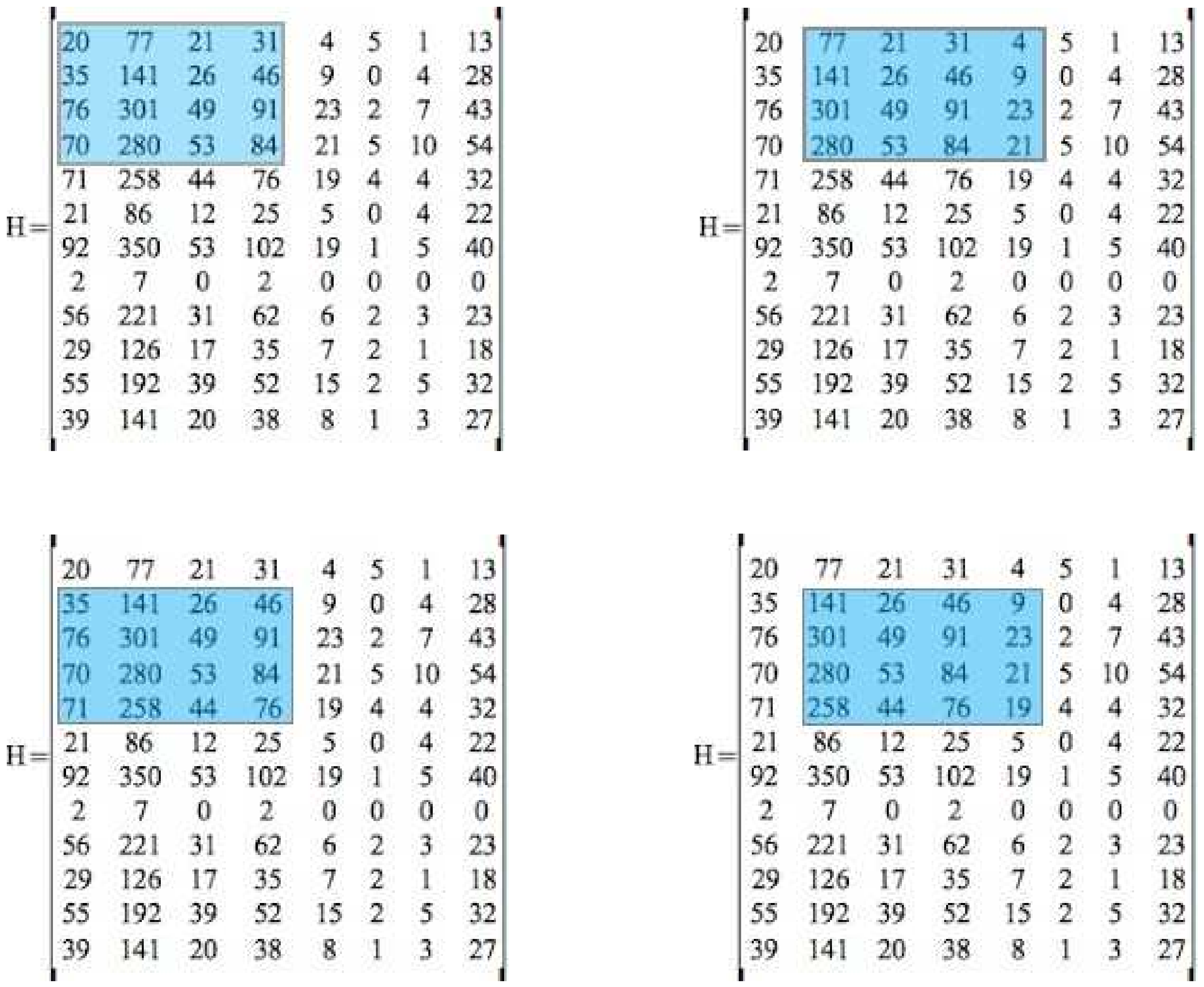

Figure 1: Window movement when shingling in images

It can be seen from the above image how window moves one by one to capture shingles of *k* X *k* dimensions. Moreover, since the images are converted into RGB format each shingle is a three-dimensional array of shape (k,k,3) where 3 is the number of channels (i.e., Red-Green-Blue). This way we can represent an image by the set of hash values of its *k*-shingles. Equivalently, each image is a 0/1 vector in the space of *k*-shingles. As each unique shingle is a dimension and hence such vectors are very sparse. And in order to save memory, these vectors can be stored in the form of bit/Boolean vectors. This results in a Boolean matrix of images where each row is a unique shingle and each column is an image.

One of the good things about shingling is that images can have different dimensions because at the end of shingling step, each image will have a set of shingles which are not required to be of the same length for all the images.

For this project, k = 16 has been chosen for good results. If k is taken too small then most images will have most shingles (false-positives). And if k is taken too large then very few images will have most shingles in common (false-negatives).

2. MIN-HASHING

Key idea of Min Hashing is to "hash" each column C to a smaller signature *h*(C), such that:

1. *h*(C) is small enough that the signature fits in memory
2. similarity(C1, C2) is the same as the "similarity" of signatures *h*(C1) and *h*(C2)

And the goal is to find a hash function *h* such that if similarity (C1,C2) is high, then we can say with high probability that h(C1) = h(C2) and vice-versa.

Since, we are dealing with shingle space in millions we take a slightly different approach than finding the index of the first (in the permuted order π) row in which column C has value 1 because taking random permutations of rows of that huge length is computationally expensive. In order to tackle that we initialize each entry of our signature matrix to ∞. And update it each entry for each column while traversing through each row and checking if that row has a 1 or not. If it has then check if the random hash function value for that signature matrix position is smaller than the current value or not. If it is we update it with the hash function value.

3. RANDOM HYPERPLANES

Random Hyperplanes method is a (d1, d2, (1-d1/$\boldsymbol{\pi}$), (1-d2/$\boldsymbol{\pi}$))-sensitive family for any d1 and d2. Key idea of this method is to pick a random vector v, which determines a hash function h_v with two buckets such that:

$$h_v(x) = +1 \text{ if } v \cdot x \geq 0 \text{ otherwise } h_v(x) = -1 \text{ if } v \cdot x < 0$$

So we pick 100 random vectors, and hash our bit vectors for each vector. And the result is a signature (sketch) of +1's and –1's for each bit vector. Since it is expensive to pick random vectors in the k-shingle space, we simply create random vectors of +1's and -1's.

Main reason why bit vector from shingling step is used instead of the data point (in our case an image) because:

1. Images are of different dimensions ranging in width from 300 pixels to 800 pixels and in height from 250 pixels to 1200 pixels. Not to mention they have a third dimension of channels (RGB).
2. Even if all of them are resized to a fixed height and width they cannot be hashed using a vector.

4. LOCALITY SENSITIVE HASHING

Key idea of locality sensitive hashing (LSH) is to hash columns of (signature or sketch) matrix several times. Matrix is divide into b bands of r rows each. Candidate column pairs are those that hash to the same bucket for $\geq$ 1 band. Here "same bucket" means "identical in that band".

Results

Following are few of the similar groups found after finding candidate images that are similar.

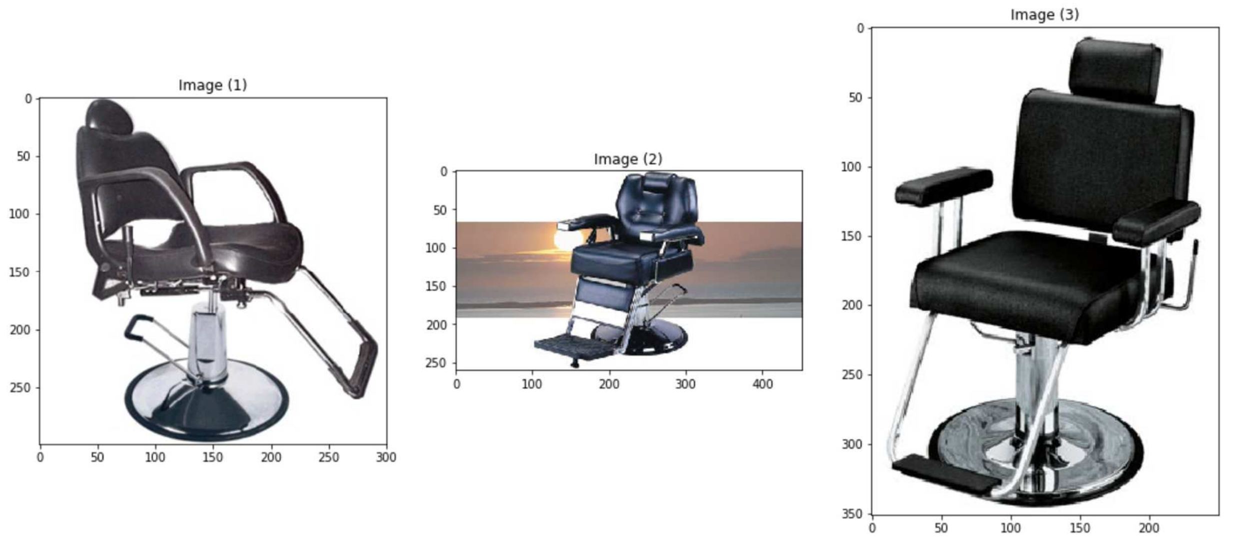

Figure 2: Similar Images of chairs

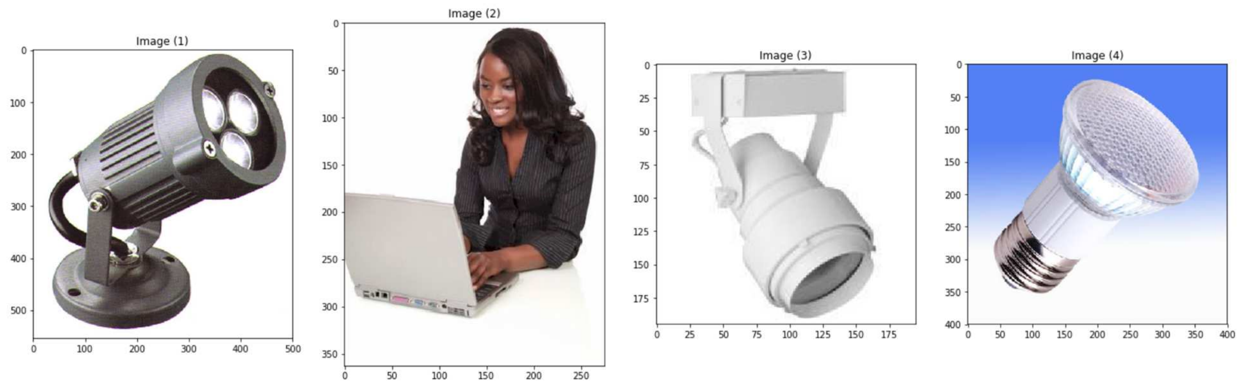

Figure 3: Similar Images of lamps with an exception in 2nd image (women working on laptop)

Conclusions

From the results produced by above mentioned method, it can be concluded that indeed Locality Sensitive Hashing technique can be applied in order to find similar images. It was observed that similar images found using Jaccard similarity (when using Min Hashing) were actually similar (Figure 2) whereas similar images found using Cosine similarity (when using Random Hyper Planes) were not all similar (Figure 3).

Moreover, when shingling in images it is important to use a shingle window of dimension (k, k, 3).

References

1. For doubts regarding implementation in SPARK and setting up AWS instances:
    a. http://www.stackoverflow.com/
    b. https://blog.insightdatascience.com/
2. Course Lecture slides for theory and idea of algorithms:

   https://cs.gmu.edu/~dbarbara/CS657/CS657L8.pdf