Sub-linear time Image search using LSH

Dinesh Avula, Ayush Jha

1. Introduction

When we have a large dataset of images and we want to find similar images to a given query image, the traditional approach of converting images to vectors may not be ideal as the complexity of images can vary greatly. To capture semantic and contextual details accurately, we use Convolutional Neural Networks (CNNs) to extract important features and patterns from the images and output a vector representation of each image. We then use nearest neighbor algorithms to find similar images to the query image. However, for large datasets, this can be computationally expensive, and to make query time faster, we use Locality Sensitive Hashing (LSH) based hashing algorithms to perform approximate nearest neighbor search in sub-linear time. This approach allows us to trade off accuracy for speed and is suitable for applications where high accuracy is not critical.
We are using a subset of the dataset image net called Tiny-image-net

In summary, we use CNN as our image processing algorithm on our dataset in order to meaningfully convert image to vectors. We use cosine similarity based LSH to hash the entries of dataset into bins.

2. Describing the algorithm

2.1. Dataset

The dataset consists of $(64 \times 64 \times 3)$ images where the number of training images are $100,000$ (1 lac) and the number of test or query images are $10,000(10\%)$ (10k). The images are from 200 different classes.

2.2. CNN

Firstly, the complexity of images can vary greatly. Each image can contain different objects, textures, colors, and shapes. In order to correctly capture the semantic and contextual details from the image, we need to use a model that can learn the relevant features from the image. Convolutional Neural Networks (CNNs) are a type of deep learning model that can learn these features. CNNs are designed to learn the relevant features and patterns from images. These networks consist of multiple convolutional layers that perform convolution operations on the input images to extract features.

2.3. LSH

- LSH or Locality Sensitive Hashing is an algorithm that is used to hash entries into bins wil some guarentees.
- The guarentees of LSH for a given tuple (r, Cr, p_1, p_2) and inputs x,y are
  - If $distance(x, y) < r$ then $Pr_{h\in H}[h(x) = h(y)] \geq p_1$ and
  - If $distance(x, y) > Cr$ then $Pr_{h\in H}[h(x) = h(y)] \leq p_2$

  where *H* is the Universal hash family and C is some constant that is greater than 1

There are multiple types of LSH that can be used some of them are

- LSH using min-hash
- LSH using sim-hash
- LSH using cosine similarity
- LSH using dot product

We are currently using cosine similarity as a similarity measure for our LSH .

We are generating a pivot that is a randomly generated vector of 1000 dimensions and normalising it to a vector of unit norm.

2.4. Describing our input for LSH

Our input is in the form of images of dimensions mentioned above. We run CNN on it to get output in the format of 1000 dimensional vectors .

We run LSH on the 1000 dimensional vectors to classify the train set into bins and store these values/positions of the images in bins.
But before we give input to our LSH we normalise our data so that the LSH can run faster. We can also normalise each entry when LSH is running on it but this causes time overhead while running the LSh.

2.5. Intuition behind algorithm

Take a look at the following image **??**

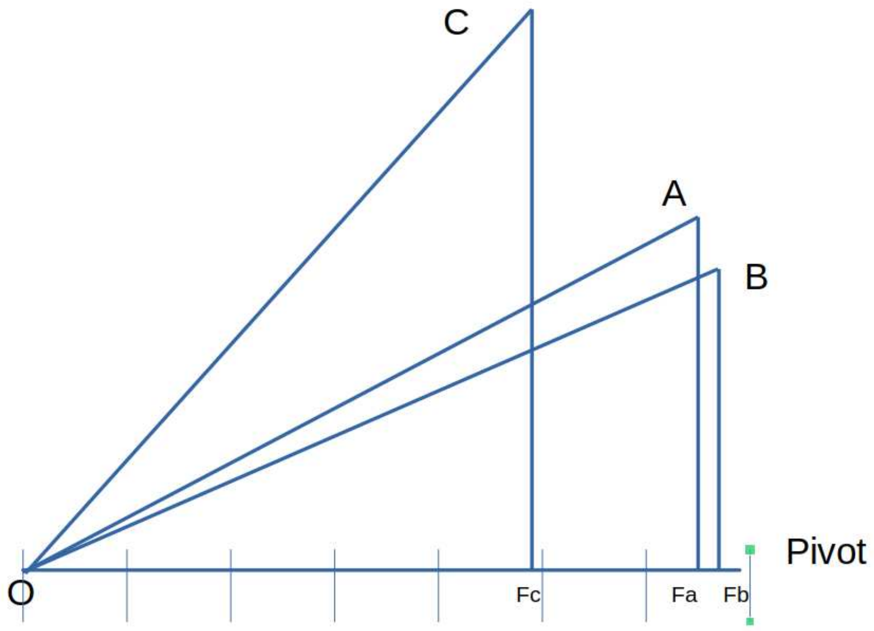


Fig. 1: Visualisation of LSH

Here all A,B,C.pivot are all unit vectors and the distances between the separators(vertical lines) on the pivot.

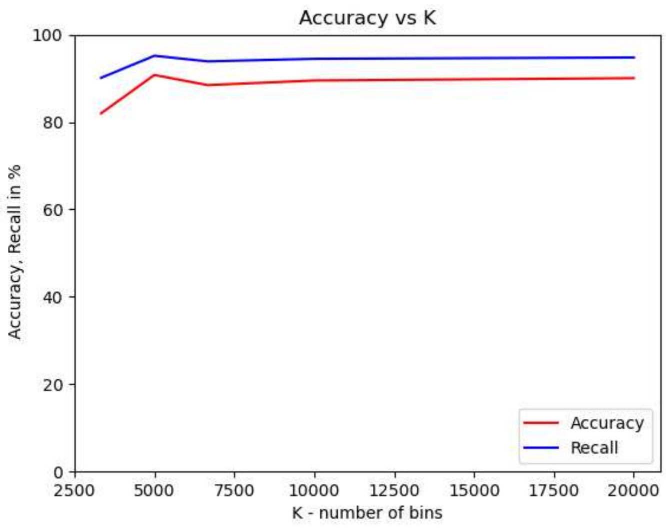
As you can see that the angle between A and B is low so both of them falls into the same bucket. In the figure we have same size of each bucket which is the space between two vertical lines on the pivot
So using this logic we hash the entries into buckets

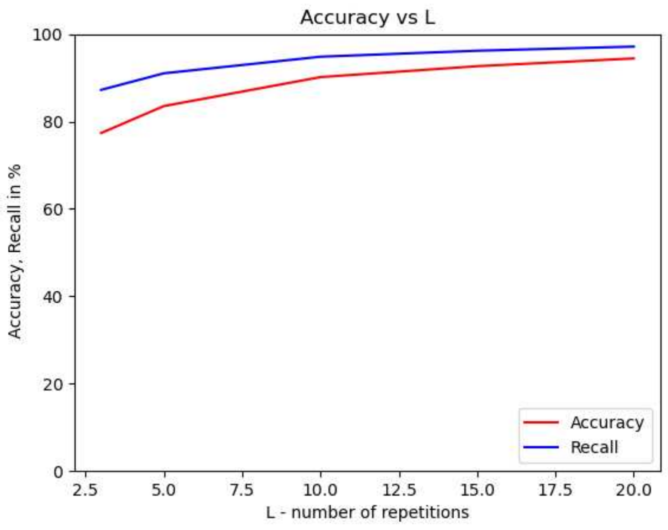
2.6. Describing our algorithm

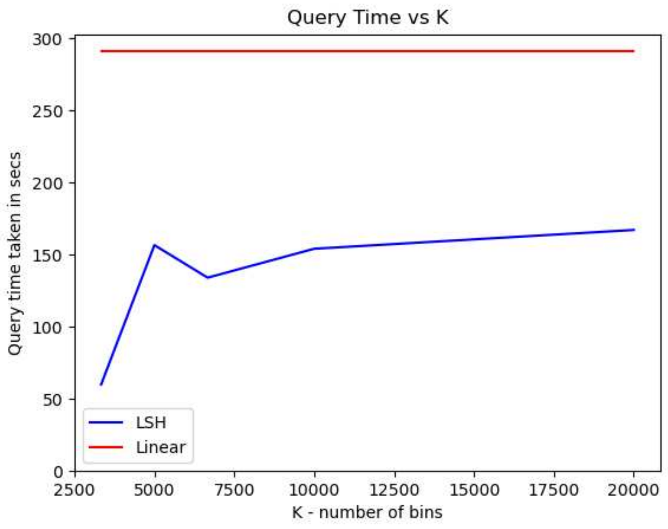
- Our LSH is taking the $1e^3$ dimensional input of size $1e^5$ and generating k hash bins .
- We are running the LSH L times so the total number of bins generated in the end are $L \times k$.
- Now when we get an query we hash the query using LSH and run our similarity algorithm on the union of all the L bins the LSH hashes to over L iterations

3. Results

We have run our algo and averaged over 5 times and generated these plots

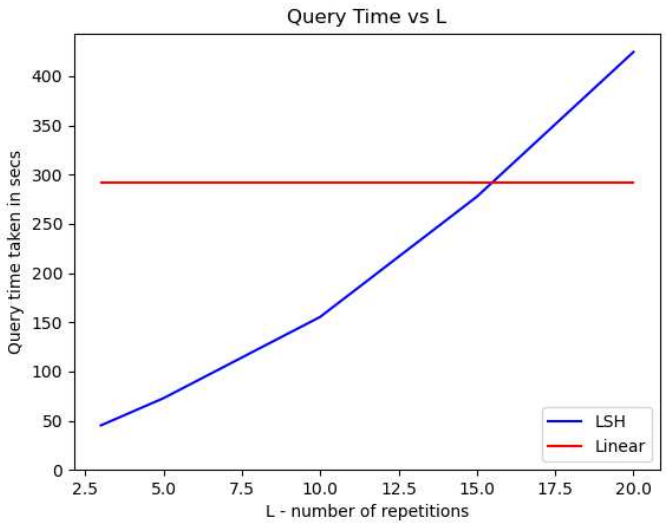


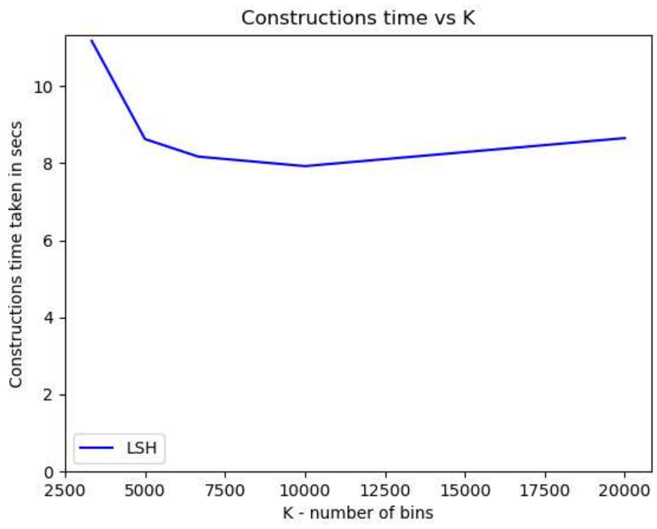


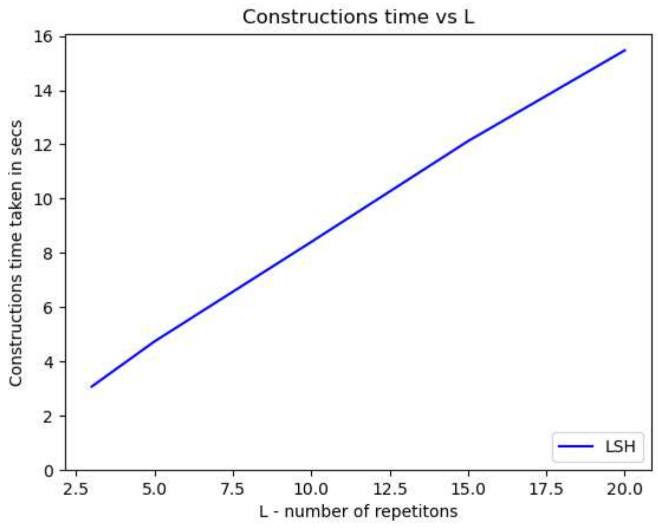


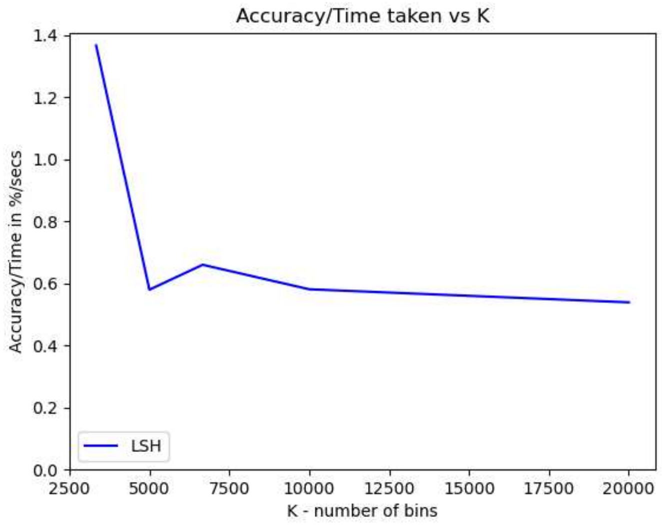
4. Conclusions

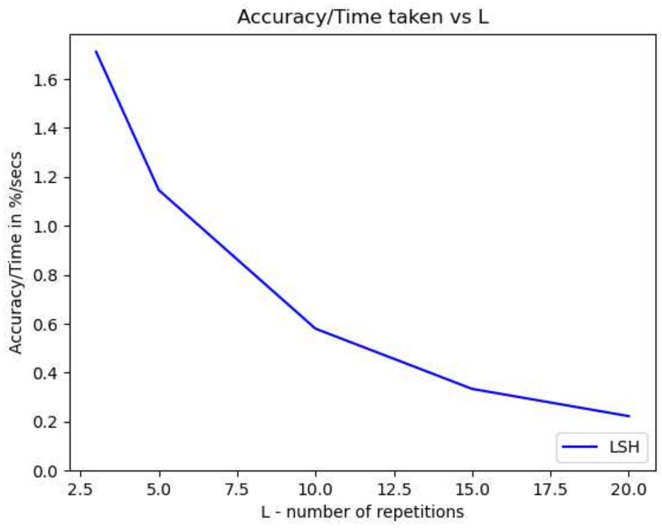
From the results we have a significant speedup in query time with a low downgrade in accuracy


Query Time vs L
Query time taken in secs
L - number of repetitions
LSH
Linear
0
50
100
150
200
250
300
350
400
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Constructions time vs K
Constructions time taken in secs
K - number of bins
LSH
0
2
4
6
8
10
2500
5000
7500
10000
12500
15000
17500
20000


Constructions time vs L
Constructions time taken in secs
L - number of repetitons
LSH
0
2
4
6
8
10
12
14
16
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Accuracy/Time taken vs K
Accuracy/Time in %/secs
K - number of bins
LSH
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
2500
5000
7500
10000
12500
15000
17500
20000


Accuracy/Time taken vs L
Accuracy/Time in %/secs
L - number of repetitions
LSH
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0