Approximate nearest neighbors by deep learning hashing on large-scale search

Michael Shell
School of Electrical and
Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332–0250
Email: http://www.michaelshell.org/contact.html

James Kirk
and Montgomery Scott
Starfleet Academy
San Francisco, California 96678–2391
Telephone: (800) 555–1212
Fax: (888) 555–1212

***Abstract*—An optimal data representation is an important task in information retrieval. A fast indexing method increases the performance level and a good representation offers precise discrimination.**

In this Similarity search algorithms based on hashing were proposed to query high-dimensional datasets due to its fast retrieval speed and low storage cost. Recent studies, promote the use of CNN! (CNN!) with hashing techniques to improve the search accuracy. However, there are challenges to solve in order to find a practical and efficient solution to index CNN features, such as the need for heavy training process to achieve accurate query results and the critical dependency on data-parameters. Aiming to overcome these issues, we propose a new method for scalable similarity search, i.e., Deep frActal based Hashing (DAsH), by computing the best data-parameters values for optimal sub-space projection exploring the correlations among CNN features attributes using fractal theory. Moreover, inspired by recent advances in CNNs, we use not only activations of lower layers which are more general-purpose but also previous knowledge of the semantic data on the latest CNN layer to improve the search accuracy. Thus, our method produces a better representation of the data space with a less computational cost for a better accuracy. This significant gain in speed and accuracy allows us to evaluate the framework on a large, realistic, and challenging set of datasets.

I. Introduction

The increasing availability of data in diverse domains has created a necessity to develop techniques and methods to discover knowledge from massive volumes of complex data, motivating many research works in databases, machine learning, and information retrieval communities. This has driven the development of scalable and efficient techniques to organize and retrieve this kind of data. Similarity search has been the traditional approach for information retrieval. Although several similarity search algorithms have been proposed to speed up similarity queries, most of them are either affected by the well-known "curse of dimensionality". Retrieve complex data causes stability problems when the data dimensionality is very high [**?**].

One of the few approaches that ensure an approximate solution with sublinear search cost for high-dimensional data is the Locality Sensitive Hashing (LSH) [**?**]. LSH is based on the idea that closeness between two objects is usually preserved by a random projection operation. In other words, if two objects are close together in their original space, then these two objects will remain close after a scalar projection operation. However, it presents some difficulties for approximate kNN queries, in particular, related to data domain parameter dependence and quality results. Therefore, in complex domains, in particular, in high dimensional data problems, an approximate solution with a solid theoretical analysis may be the best option in many application areas because of their efficiency in time and space.

On the other hand, in Machine Learning traditionally images are often described by the hand-craft visual features. However, these hand-craft features cannot well reveal the high-level semantic meaning (labels or tags) of images, and often limit the performance of image retrieval [**?**]. Inspired by recent advances in **CNN!** (**CNN!**) [**?**], many methods solved the problem of precision of similarity retrieval by using CNN as feature extractor and then build a compact similarity-preserving hash code for fast image retrieval. Again, hashing is widely used for large-scale image retrieval as well as video and document searches because the compact representation of hash code is essential for data storage and reasonable for query searches [**?**]. However, some drawbacks based on these supervised hashing methods have not been solved entirely, as follows:

- There is a trade-off between classification error and quantization error: activations of lower layers are more general-purpose [**?**], so training is more effective. However lower layers have larger activations maps (many nodes), which are harder to encode which leads to a compromise.
- There is a dependency on parameter values for approximate similarity search schemes based on LSH, which determine the number of hash functions and number of hash tables.

This paper proposes a novel supervised hashing technique, named Deep frActal based Hashing (DAsH), designed to perform scalable approximate similarity search. The contributions of our work are as follows. First, we introduce and define a scheme based on CNN and optimized using fractal theory. To overcome the limitation of large activations on lower layers of CNN (output of the last convolutional layer) we reduce its dimensionality using autoencoders to the optimal sub-space. Then we index this new representation with LSH scheme. Second, we present a novel method, based on fractal theory, which allow us to can find the optimal number of hash functions for an approximate similarity search scheme based on LSH.

The paper is organized as follows. Section 2 summarizes the background for this work. Section 3 describes the proposed technique and Section 4 reports experimental results on real and synthetic datasets. Finally, we conclude in Section 5.

II. Locality Sensitive Hashing

Previous work [?] has explored the idea of hashing objects and grouping them into buckets with the goal of performing approximate similarity search within buckets associated with the query element. The idea behind LSH is that if two objects are close together in their original space, then these two objects will remain close after a scalar projection operation. Hence, let $h(x)$ be a hash function that maps a d-dimensional point x to a one-dimensional value. The function $h(x)$ is said to be *locality sensitive* if the probability of mapping two d-dimensional points x_1, x_2 to the same value grows as their distance $d(x_1,x_2)$ decreases.

LSH based methods report efficient results when adequate values for m (number of hash functions) and L (number of indexes) are chosen. The E^2-LSH algorithm find the best value for m and L by experimentally evaluating the cost of calculation for samples in the given dataset. Basically, the tuning parameter of LSH is chosen as a function of the dataset to minimize the running time of a query while the space requirement is within the memory bounds [?].

A. Fractal Theory

A fractal is characterized by the self-similarity property, i.e., it is an object that presents roughly the same characteristics when analyzed over a broad range of scales [?]. From the Fractal Theory, the Correlation Fractal Dimension $\mathfrak{D}$ is particularly useful for data analysis, since it can be applied to estimate the intrinsic dimension of real datasets that exhibit fractal behavior, i.e., exactly or statistically self-similar datasets [?]. It has been shown that, given a set of N objects in a dataset with a distance function $d(x,y)$, the average number of k neighbors within a given distance r is proportional to r raised to $\mathfrak{D}$. Thus, the pair-count $PC(r)$ of pairs of elements within distance r follows the power law:

$$PC(r) = K_p \times r^{\mathfrak{D}} \tag{1}$$

where, K_p is a proportionality constant, and $\mathfrak{D}$ is the correlation fractal dimension of the dataset. Consequently, a fractal is defined by the self-similarity property, that is the main characteristic that represents exactly or statistically the similarity between the parts to the whole fractal.

III. Deep Fractal based Hashing - DAsH

In this section, we propose the Deep Fractal based Hashing (DAsH) designed to perform a scalable approximate search by supervised hashing by a supervised hashing scheme. As introduced in Section 1, our strategy is to use the fractal theory to find the optimal sub-space for the last convolutional layer output of the CNN network, and the optimal number of hash functions for LSH index as well.

Figure **??** illustrates the training process structure. The network consists of three types of layers: 1) convolutional layers which weights are pre-trained most of the time on Imagenet and the target dataset is fine-tuned via transfer learning; 2) fully connected layers with the last softmax layer which return the categorical probability distribution; 3) autoencoders layers which are used for dimensionality reduction. The **CNN!** (**CNN!**) is trained end-to-end with the groundtruth labels. We use the output of the last convolutional layer because it has the most general-purpose representation for learning, but it has a drawback with a high dimensional representation. To overcome the high-dimensionality problem, we reduce to the optimal sub-space using an autoencoder. Then we index the optimal sub-space obtained by the autoencoder with LSH scheme which, as we mentioned, it is also tuned thank to fractal theory. At the same time, we use another n-autoencoders to learning the representation of each class. After, we will use these autoencoders to improve the retrieval process.

As it was showed in [?] that a successful dimensionality reduction algorithm projects the data into a feature space with dimensionality close to the fractal dimensionality (FD) of the data in the original space and preserves topological properties. Thus, to find the target dimensionality (m) needed by autoencoder networks we follow the following heuristic. We start with the value at $m_1 = 2^2$, compute the FD of the new space with just that, then increment value at $m_2 = 2^3$, recompute the FD, and continue doing this until some t ($m_t = 2^t$) where we can see a flattening in the fractal dimension, meaning that more features do not change the fractal dimensionality of the dataset.

The second step of our procedure is image retrieval via DAsH. We process the query image forwarding it through CNN aiming to obtain the strongest n classes. In contrast to existing similarity learning algorithms that learn similarity from the low-level feature, our similarity is the combination of semantic-level and hashing-level similarity. So, the semantic level similarity is computed firstly. After the semantic relevance checking, we will obtain the new queries $(q_1, q_2, ...q_n)$ using the strongest n autoencoders. The query is transformed into new query objects $(q_1, q_2, ...q_n)$ which are hashed to locate the appropriate buckets. Once the buckets are located, the relevant candidate set is formed.

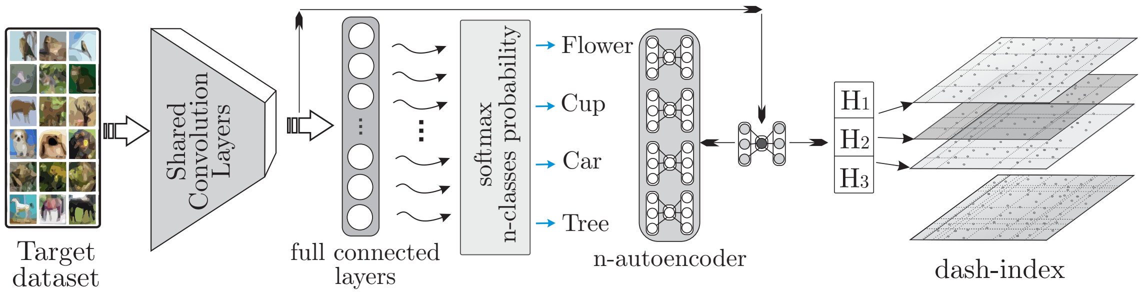


Figure 1. DAsH: Training and indexing process.

Then, the elements in the candidate set are exhaustively analyzed to recover only the objects that satisfy the query condition. This process is performed for each of the L hash tables. This process is illustrated in Figure **??**.

A. *Using Fractals to estimate LSH parameters*

To tune the LSH parameters we used a property of the correlation fractal dimension $\mathfrak{D}$, which can describes statistically a dataset. Moreover, the correlation fractal dimension $\mathfrak{D}$ can be estimated in linear time as it is depicted in [**?**].

We are interested to find out the resolution scale $log(r)$ at which there are approximately k objects. Considering the line with slope $\mathfrak{D}$ passing at a point defined as $< log(r), log(Pairs(k)) >$ the constant K_d using the Equation **??** is:

$$log(PC(r)) = \mathfrak{D} \times log(r) + K_p$$
$$K_p = log(Pairs(k)) - \mathfrak{D} \times log(r) \tag{2}$$

Considering another point $< log(R), log(Pairs(N)) >$, the constant K_d is defined as :

$$K_d = log(Pairs(N)) - \mathfrak{D} \cdot log(R) \tag{3}$$

Now, combining Equations **??**, **??**, we can define the radius r as:

$$r = R \cdot exp(\frac{log(Pairs(k)) - log(Pairs(N))}{\mathfrak{D}}) \tag{4}$$

Using the last equation **??** we find out that the optimal number of hash functions m for a **LSH!** (**LSH!**) based index configured to retrieve the k nearest neighbors is proportional to the number of pairs at a distance r. This has sense, because an average number of k neighbors are within a given distance r. Then, we define:

$$m \approx log(PC(r)) \tag{5}$$

combining equations **??** and **??** we will obtain that $m \approx \mathfrak{D} \cdot log(r)$. Experimentally, we confirm out that the optimal m is:

$$m = (\lceil \mathfrak{D} + 1 \rceil) \cdot log(r) \tag{6}$$

IV. SUPERVISED HASHING: A SIMPLE BASELINE

In this section, we describe the protocols used in the literature for SSH and SH and explain how a simple strategy efficiently solves the similar problems.

A. *Evaluation protocols of SSH and SH*

SSH consists in index a dataset of N images $\mathcal{I}_{train}$, of which a subset $\mathcal{I}_{label} \subseteq \mathcal{I}_{train}$ is labeled. But, SH is the extreme case $\mathcal{I}_{label} = \mathcal{I}_{train}$. If we have an unlabeled query image q, the system have to return an ordered list of images from the $\mathcal{I}_{train}$. To evaluate, we have a dataset of queries; the evaluator knows all labels of the queries as well as the labels in $\mathcal{I}_{train}$, also in the SSH setting, then an image is considered correct if it and query have the same label. The performance is measured by precision or mean average precision (mAP).

Then, given a query q, if the i^{th} image is correct for q we define $\delta(q,i) = 1$, and 0 otherwise. The precision at (rank) k is given by $P(q,k) = \frac{1}{k}\sum_{i=1}^{k}\delta(q,i)$. Denoting by $cl(q) = \sum_{i=1}^{N}\delta(q,i)$ the total number of correct images in $\mathcal{I}_{train}$, and the average precision at k is $AP(q,k) = \frac{1}{cl(q)}\sum_{i=1}^{k}\delta(q,i)P(q,i)$. The mAP at k (or simply mAP when $k = N$) is the mean AP over all test queries.

B. *Retrieval through class probability estimation*

The information recovery [**?**] and learning to rank that the best prediction for precision at k is given by placing items $x \in \mathcal{I}_{train}$ n according to their probability of being correct for the query. This result extends to the optimization of mAP.

Optimal ranking for SH. In the particular setup of SH where the system knows the labels of the images in $\mathcal{I}_{train}$, the probability that a picture x with label y is correct is the likelihood $\mathbb{P}(y \mid q)$ that the query image has label y. The important point in this part is that the probability of x being correct for q only depends on the label of x. Thus, ordering the C labels so that $\mathbb{P}(y \mid q) \geq \cdots \geq \mathbb{P}(c_C \mid q)$, the best ranking is to return the complete dataset of $\mathcal{I}_{train}n$ with label c_1 first, followed by all the pictures with label c_2, and so on.

In practice, $\mathbb{P}(. \mid q)$ is unknown, but we can train a classifier on $\mathcal{I}_{label} = \mathcal{I}_{train}$ which outputs probability estimates

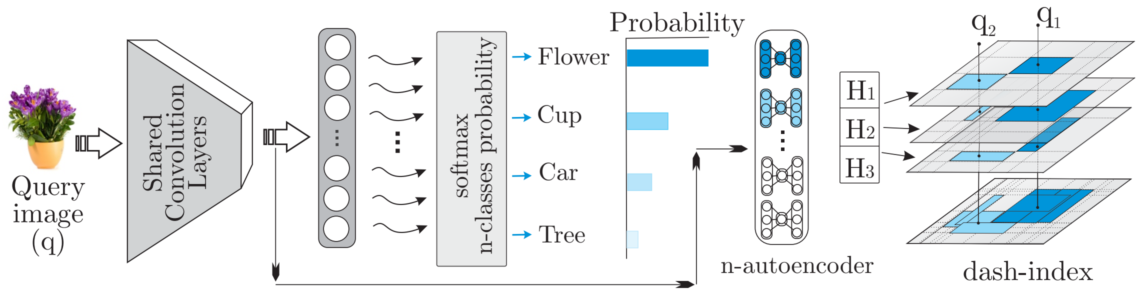


Figure 2. DAsH. Retrieval process.

$\hat{\mathbb{P}}(c \mid q)$ for every label c, and compute the optimal ranking according to $\hat{\mathbb{P}}(c \mid q)$ Such probability estimates are given by, e.g., multiclass logistic regression or a Convolutional Neural Network (CNN) with a softmax output layer. Labels of $\mathcal{I}_{train}$ are stored on $[log2(C)]$ e bits or in an inverted file.

In practice, $\mathbb{P}(. \mid q)$ is unfamiliar, but we can train a classifier on $\mathcal{I}_{label} = \mathcal{I}_{train}$ which outputs probability con predict $\hat{\mathbb{P}}(c \mid q)$ for every label c, and process the optimal ranking according to $\hat{\mathbb{P}}(c \mid q)$ Such probability estimates are shown by, e.g., multiclass logistic regression or a Convolutional Neural Network (CNN) with a softmax output layer. Labels of $\mathcal{I}_{train}$ are stored on $[log2(C)]$ e bits or in an inverted file

Relationship between classification accuracy and ranking performance. If we show the classification accuracy as p, then the resulting mAP is at least p. When the classifier foretells the class of q properly, all images of that class will be ranked first, and the resulting AP(q) is 1; this happens on a proportion p of the queries. So, the classification accuracy is a lower bound on the mAP.

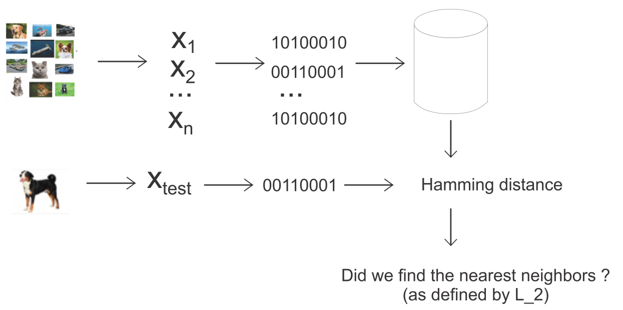
Optimal ranking for SSH. In the more general setup of SSH, we do not know the label of some images in $\mathcal{I}_{train}$. Yet, considering the (real) probabilities $\mathbb{P}(c \mid q)$ and $\mathbb{P}(c \mid x)$, the probability that x is correct for q is given by $\sum_{c=1}^{C} \mathbb{P}(c \mid q)\mathbb{P}(c \mid x)$ it is the probability that both q and x have the same label, assuming conditional self-sufficiency of the labels of the query and the image. Notice that this is the dot product between the dependent label probability vectors of q and x.$Then, given probability measures for the labels of queries and im$ we consider two retrieval algorithms:

- **Classifier topline:** : For each image $x$of $\mathcal{I}_{train}$, saves a vector $u(x)$ equal to either (1) the one-hot encoding vector of the label of x if $x \in \mathcal{I}_{label}$, or (2) the full vector $\hat{\mathbb{P}}(. \mid x)$). Rank images x according to the dot product $\langle \hat{\mathbb{P}}(. \mid q), u(x) \rangle$ This strategy corresponds to the best strategy, but needs to save the probability vectors for each image in $\mathcal{I}_{train} \backslash \mathcal{I}_{label}$
- **Classifier hashed:** Here we hash the probability vector. The first hashing method that we evaluate, is the one-hot strategy, which saves the index of the maximal activation on $\lceil log_2(C) \rceil$ bits. This approach, expressed **Classifier+one-hot** in what results, returns all images of the strongest class first. The second encoding, referred to as Classifier+LSH, is locality-sensitive hashing (LSH) with tight frames [**?**], a simple non data adaptive hashing scheme. This LSH method provides binary vectors that are compared with Hamming distances. Consequently it can be used as drop-in replacements for the competing binary encoding methods.

V. How should we evaluate supervised hashing?

A. *HASHING FOR NEAREST NEIGHBOR SEARCH*

Compress a set of vectors $(x_i)_{i=1}^{b}, x_i \in \mathbb{R}^d$



B. *SUPERVISED HASHING*

Compress a set of vectors and their labels $((x_i, y_2))_{i=1}^{n}, x_1 \in \mathbb{R}^d, y_1 \in \{1, ..., L\}$

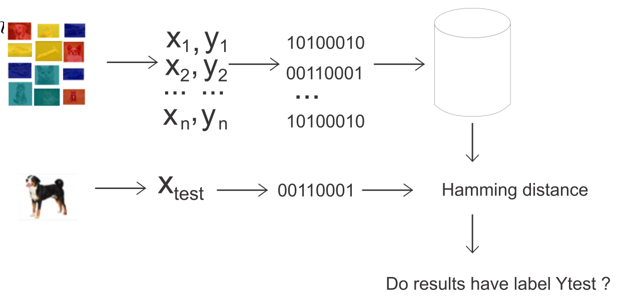


Figure 3. HASHING FOR NEAREST NEIGHBOR SEARCH

Table I
Results on CIFAR-10

Features	n_label	$n_anchors$	Method	bits	mAP
GIST	59,000	1,000	SQ[1]	64	0.704
			SQ[1]	128	0.712
			One-hot	4	0.762
GIST	5,000	1,000	SDH[3]	64	0.402
			One-hot	4	0.377
			LSH	64	0.430
			Topline	-	0.578
GIST	1,000	300	KSH[4]	12	0.232
			KSH[4]	48	0.284
			One-hot	4	0.270
			LSH	48	0.309
			Topline	-	0.350
DEEP	50,000	-	⋮	⋮	⋮
AlexNet			DSH[2]	12	0.616
			DSH[2]	48	0.621
			One-hot	4	0.870

Table II
Results on ImageNet

Features	Method	bits	mAP@1500
VGG	SQ[1]	128	0.620
VGG	one-hot	10	0.664

- Supervised hashing [1,2]: labels y known for all x in the reference set
- Semi-supervised hashing [3,4]: labels y known for only n_label samples

C. Why not just encode the class ID?

Supervised hashing with classification baseline

- Trivial binary encoding of the class id,$e.g. y = 9 \rightarrow 1001$

$$x_1, y_1 \quad 1001 \tag{7}$$
$$x_2, y_2 \quad \rightarrow \quad 1001 \tag{8}$$
$$\dots \quad \dots \tag{9}$$
$$x_n, y_n \quad 0011 \tag{10}$$

- Train classifier on pairs $(x_1, y_1), \dots, (x_n; y_n)$ and predict y_test with the classifier
- Guaranteed performance: $mAP \geq$ classifier accuracy

D. Extension to semi-supervised hashing

$$\begin{aligned}\mathbb{P}(x\, correct for\ q) &= \sum_{j=label} \mathbb{P}(j \mid x)\mathbb{P}(j \mid q) \\ &= (\mathbb{P}(j \mid x), \mathbb{P}(j \mid q)) \\ &= (\underbrace{\hat{\mathbb{P}}(j \mid x)}_{classifier}, \hat{\mathbb{P}}(j \mid q))\end{aligned} \tag{11}$$

- Train $\hat{\mathbb{P}}$ on labelled images
- Compute $\hat{\mathbb{P}}(\cdot \mid x) \in [0,1]^L$ for x unlabelled
- Compress $\hat{\mathbb{P}}(\cdot \mid x)$ with one-hot / LSH

Encoding schemes

- One-hot:$(0.15, 0.07, 0.08, \mathbf{0.7})30011$
- LSH:$(0.15, 0.07, 0.08, 0.7)(sign(w_i^T \cdot v + b_i))_{i=1}^{n_bits}$
- Top line: no compression

E. How can we avoid this biased protocol?

??.

- Test on classes **never seen at train time** classes in 4 folds, each with 25% of classes,Both setups:
  - Train hash functions on train75
  - Encode train25 with hash functions
- **Setup 1: Retrieval with hash codes**
  - Use train25 as reference set
  - Use test25 as queries
- **Setup 2: Classification on hash codes**
  - Train classifier using train25 labels
  - Evaluate accuracy on test25

F. Unsupervised baseline for proposed protocol

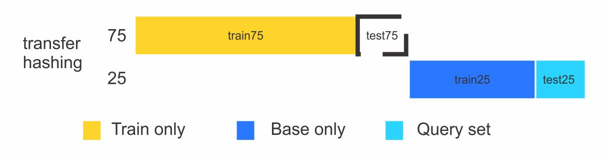
- **Experimental setting**
  - Experiments with CIFAR-10 using AlexNet
  - Unsupervised PQ codes [5] with 4 bytes
- **Setup 1: Retrieval with hash codes**
  - PQ codes with asymmetric comparison
  - Higher layers are better
  - 4 bytes enough for most of performance
  - Inner product on softmax gets the best result
- **Setup 2: Classification on hash codes**
  - Drop in accuracy due to encoding
  - Lower layers more generic→better accuracy
  - Lower layers high dimensional→larger gap between PQ and full vector
  - → Trade-off encoding/accuracy

VI. Evaluation Protocols

In this section, we describe the two evaluation tasks proposed in [?], namely retrieval of unseen classes, and transfer learning to new classes. They correspond to application cases on large datasets. The two protocols differ only in the evaluation metric: ranking versus class accuracy.

Dataset definition. at test time we use separate classes from a standard classification dataset. When learning the hashing function 75% of the classes are assumed to be known, and the 25% remaining classes are used to evaluate the encoding/hashing scheme. We call train75/test75 the train/test images of the 75% classes and train25/test25 the remaining ones.

Protocol 1: Retrieval of unseen classes, to index train25 and use test25 as queries, we use the hashing scheme. We use the labels of train25 for evaluation only. This setup is like an instance search approach except that the class labels give the ground-truth. The train75 - train25



- test25 split is the supervised equivalent of the learn - database - query split in unsupervised hashing.

Protocol 2: Transfer learning, the authors proposed to train a new CNN from scratch with the same structure as the top of the original network, using the stored train25 descriptors. The goal is to maximize the transfer accuracy on test25.

VII. Experiments

In this section, we are interested in answering the following question: (a) How accurate is our model in estimating the LSH parameters using the fractal dimension; (b) How does our DAsH method improve the other LSH implementations in terms of *querying performance* and *precision*. The performance of DAsH method was compared to two well-known approximate search methods, namely Multi-probe LSH [?], LSH-Forest [?], ITQ [?], and LOPQ [?]. All of the experiments were performed on a workstation with Intel core i7 3.0Ghz (12 cores) CPU and 64Gb RAM which is supplied with four Geforce GTX 1080 GPU with 8Gb VRAM each one.

We first conduct experiments on eight widely used datasets using hand-crafted features (AUDIO, CITIES, EIGENFACES, HISTOGRAMS, MGCOUNTY, RANDOMWALK, SYNTH16D, SYNTH6D, VIDEO)[1] to evaluate our proposed method for estimating the LSH parameters. Beside hand-crafted features, we also show the effectiveness of our methods when deep features are extracted by the deep Convolutional Neural Networks (CNN), we conduct this experiment on three datasets (MNIST[2], CIFAR-10[3], SVHN[4]) to evaluate our in terms of querying performance, meap average precision (mAP), and precision. The following describes the details of the experiments and results.

A. Experiment 1: Tunning LSH Parameters

LSH based methods report efficient results when adequate values for m (number of hash functions) are chosen. To evaluate the effectiveness of the presented approach to tune the LSH parameters using fractal dimension, we worked on a variety of synthetic and real dataset. Table **??** summarizes the main features and parameters of the datasets, including the number of elements N, number of attributes d, their intrinsic (fractal) dimension $\mathfrak{D}$, the LSH parameters computed using two approaches: the Andoni [5] algorithm and our proposal based on fractal dimension, and the total computation time for tune the LSH index (in seconds). The experiment results for the number of hash functions m show that the estimations given by Equation **??** are comparable with those obtained with the E2LSH algorithm proposed by Andoni using up to $10X$ less time.

B. Retrieval Performance

The aim of this experiment is to measure the total time spent retrieving the k-nearest neighbor objects. The data structures being compared were tested with an specific values for queries. Thus we use $k = 1000$ when compute the Mean Average Precision(mAP) metric and $k = 25$ when compute the precision metric $(P(\%))$.

Table **??** show the comparison in terms of mean average precision mAP, precision, and total time (in seconds).

VIII. Conclusions

In this paper, we presented a new scheme to solve approximate similarity search by supervised hashing called Deep Fractal base Hashing. Our approach shows the potential of boosting querying operations when a specialized index structure is designed from end-to-end. Due to the abilities of Fractal theory to find the optimal sub-space of the dataset and the optimal number of hash functions for LSH Index, we are able to find an optimal configuration for learning and indexing process. Moreover, we defined a novel method, based on fractal theory, which allows us to can find the optimal number of hash functions for LSH index. We can estimate these parameters in linear time due to it depends on computing fractal dimension.

We conducted performance studies on many real and synthetic datasets. The empirical results for LSH parameters show that our method based on fractal theory is comparable with those obtained with the brute-force algorithm using up to $10X$ less time. Moreover, in retrieval performance, the DAsH method was significantly better than other approximate methods, providing up to 8% better precision maintaining excellent retrieval times.

[1]https://github.com/joselhuillca/fractal_dataset
[2]http://yann.lecun.com/exdb/mnist/
[3]https://www.cs.toronto.edu/~kriz/cifar.html
[4]http://ufldl.stanford.edu/housenumbers/

[5]http://www.mit.edu/~andoni/LSH/

Table III
Optimal LSH Params using exhaustive e2lsh and the fractal based method.

	dataset		fractal params			e2lsh		fractalsh		
	N	d	$\mathfrak{D}$	$log(R)$	$log(CR)$	m	$time(s)$	r	m	$time(s)$
audio	54387	192	6.49	-1.30	17.00	18	64.20	0.14	16	12.68
cities	5507	2	2.36	2.30	16.00	8	16.45	4.16	6	0.81
eigenfaces	11900	16	4.25	-1.40	18.00	12	42.28	0.13	13	1.57
histograms	4247	256	2.50	-0.81	16.01	6	15.62	0.22	7	6.12
mgcounty	27282	2	1.81	0.70	19.00	4	87.96	0.27	4	1.13
randomwalk	10000	1024	5.52	2.80	15.00	16	50.60	10.16	17	10.60
synth16d	10000	16	8.36	-1.40	17.00	20	27.20	0.18	18	7.51
synth6d	10000	6	4.95	-1.40	17.00	12	28.16	0.14	12	6.47
video	79094	50	7.73	-1.40	21.00	16	1205.73	0.13	19	92.54

Table IV
Mean Average Precision(mAP), precision, and cumulative time spent to compute mAP for different methods on the MNIST, SVHN and CIFAR-10 datasets.

	MNIST			**SVHN**			**CIFAR-10**		
	mAP	**P (%)**	**time(s)**	**mAP**	**P (%)**	**time(s)**	**mAP**	**P (%)**	**time(s)**
MpLSH [?]	0.86	0.95	19.58	0.71	0.73	8.09	0.83	0.89	122.45
ITQ [?]	0.87	0.95	1297.32	0.70	0.80	8411.26	0.84	0.88	1056.83
LOPQ [?]	0.86	0.90	119.21	0.71	0.74	613.65	0.83	0.88	528.20
LSH-F [?]	0.85	0.96	217.33	0.61	0.78	365.90	0.86	0.89	365.47
DAsH (Ours)	0.93	0.98	20.82	0.74	0.84	10.34	0.88	0.91	125.37

Table V
My caption

	SMALL SPACE						**ORIG. SPACE**
AGNEW							
Dim	16	32	64	128	256	512	8704
Fractal Dim	4.2656	4.5797	5.7044	20.5952	24.7651	24.7651	30.4943
CIFAR10							
Dim	16	32	64	128	256	512	4096
Fractal Dim	1.4436	5.4327	7.2287	23.2534	25.5753	25.5753	1.6494
ISBI							
Dim	16	32	64				4096
Fractal Dim	11.6047	16.1283	16.1283				16.1283
MNIST							
Dim	16	32	64	128	256		800
Fractal Dim	4.3912	4.1542	22.5753	22.5753	22.5753		22.5753
SVHN							
Dim	16	32	64	128			1152
Fractal Dim	6.5841	4.6553	28.3359	28.3359			28.3359