Revision to University of Minnesota CS dept. Tech. Report available at
http://www.cs.umn.edu/tech_reports_upload/tr2011/11-028.pdf

Efficient Similarity Search via Sparse Coding

Anoop Cherian Vassilios Morellas Nikolaos Papanikolopoulos
{cherian, morellas, npapas}@cs.umn.edu

Abstract

This work presents a new indexing method using sparse coding for fast approximate Nearest Neighbors (NN) on high dimensional image data. To begin with we sparse code the data using a learned basis dictionary and an index of the dictionary's support set is next used to generate one compact identifier for each data point. As basis combinations increase exponentially with an increasing support set, each data point is likely to get a unique identifier that can be used to index a hash table for fast NN operations. When dealing with real world data, the identifiers corresponding to the query point and the *true* nearest neighbors in the database seldom match exactly (due to image noise, distortion, etc.). To accommodate these near matches, we propose a novel extension of the framework that utilizes the regularization path of the LASSO formulation to create robust hash codes. Experiments are conducted on large datasets and demonstrate that our algorithm rivals state-of-the-art NN techniques in search time, accuracy and memory usage.

1 Introduction

The majority of the vision tasks in the recent times require efficient strategies to search for Nearest Neighbors (NN) of a query point in a database. Examples include, but not limited to face recognition, object tracking, multiview 3D reconstruction [46] and video search engines [45]. Image data are generally high dimensional; Scale Invariant Feature Transforms (SIFT) [32], Generalized image descriptors [48], shape contexts [40], Histogram of Oriented Gradients (HOG) [9], etc. are a few examples. When the data is high dimensional, sophisticated data structures might need to be designed to make the computation of NNs efficient. The problem is not restricted to computer vision, but manifests itself in many other domains, such as document search, content-based information retrieval [11], multimedia databases [3], etc.

In this paper, we develop algorithms for fast NN operations. Our approach is motivated by recent advances in the theories of sparse signal processing and compressive sensing. The latter deals with the idea of sampling and reconstructing signals that are sparse in a specific overcomplete basis, such that perfect reconstruction can be achieved via only very few samples (as compared to the number of samples prescribed by the Shannon's sampling theorem) [12]. For general data descriptors (e.g. SIFT descriptors), it is non-trivial to find this overcomplete basis dictionary. To meet this challenge, dictionary learning techniques have been suggested [41, 16] whereby the data itself steers a basis dictionary by solving an L1 regularized least-squares problem (namely LASSO). This idea of dictionary learning has been leveraged for very successful applications such as image denoising, object classification, recognition [35], etc. A natural next question however is whether using the sparsity itself can achieve fast NN retrieval on high dimensional data. This paper examines this possibility in detail and proposes algorithms applicable to the visual data domain.

Before proceeding further, we briefly list the primary contributions of this paper:

- We propose a novel tuple representation, *Subspace Combination Tuple* (SCT), for high dimensional data using a learned dictionary that assists in indexing the data vectors through a hashtable.
- Utilizing the regularization path of the LARS algorithm for solving the LASSO formulation, we propose a simple and effective method for generating robust hash codes on the above SCTs for fast and accurate NN retrieval.

2 Related Work

Low dimensional data has several data structure based schemes that for efficient ANN retrieval, such as k-d trees, R-trees, etc.([28, 8]). Unfortunately, when the data dimensionality increases as high as 20, the efficiency of such schemes starts

dropping as shown in [2]. It is seen that in high dimensional spaces, the computational expense of these classical algorithms is no better than a linear scan over the entire dataset ([22]). Of the several schemes advocated for efficient ANN, the most popular has been *Locality Sensitive Hashing* (LSH) ([10]), which introduces the traditional hashing schemes into a probabilistic framework in which similar data points have a greater probability of colliding to the same hash bucket. It is often seen that the hash family selected for LSH do not have any connection to the structural properties of the original data, whose knowledge might help in generating smaller hash codes. Towards this end, Restricted Boltzmann Machines (RBMs) ([43]) and adaptations of Boosting ([44]) have been proposed. These algorithms need to store the original data points in the hash table to resolve collisions; hindering their scalability.

Prioritizing low memory footprint, while sacrificing retrieval accuracy, resulted in the design of optimal hash codes that capture the data properties. An effort in this direction is the well-known Spectral Hashing (SH) algorithm ([50]), which establishes a connection between efficient graph partitioning and optimal binary codes. The algorithm minimizes the sum of the Hamming distances between pairs of input data points weighted by a Gaussian kernel constructed on the respective data features. In [42], a shift invariant kernel (based on random Fourier features) is proposed instead of the Gaussian kernel, showcasing better performance. The dependence of SH on a particular kernel is avoided in [31] by estimating a low dimensional data distribution. Learning such low dimensional structures has also been adopted in other hashing algorithms such as [23]. There have been efforts to elevate standard machine learning techniques to address ANN via LSH. In [29], random linear data projections are extended to non-linear functions through a kernel mapping, the subsequent hashing algorithm referred to as Kernelized LSH (KLSH). A general trend in all these methods is the selection of a set of random hyperplanes to which the data is projected, later these projections are used for hashing in the binary Hamming space. It is often seen that the selection of these hyperplanes has a tremendous impact on the accuracy of the respective algorithm. More recently, quantization techniques have been extended to produce efficient indexing techniques in [24], in which a data vector is mapped to the indices of a Cartesian product of subspaces; the basis for each subspace is learned independently. There could be redundancies in the basis between subspaces, learning which could help efficient indexing.

Building similarity metrics on sparse codes has been investigated several times in the recent past ([27, 7]). Unfortunately, the adequacy of these metrics for retrieval problems is not thoroughly investigated. Sparse coding for image classification has been suggested in [51] and [49]. A sparse coding framework that approximates the inner-product distance between data points for retrieval is presented in [53]. All these papers agree that sparse coding is sensitive to variations in the original data vectors; as a result, applying these approaches to retrieval problems is difficult. In this paper, we show that sparse coding can in fact produce efficient and accurate ANN if used correctly. To this end, we propose a novel encoding scheme on the sparse codes that approximates the Euclidean distance between the data points. Utilizing the regularization path, we show that robust sparse codes can be produced easily. We show extensive experiments to demonstrate that our method outperforms the state of the art in retrieval against accuracy, scalability and robustness.

3 Preliminaries

3.1 Sparse Coding and Dictionary Learning

Given a data vector $v \in \mathbb{R}^d$, assume that we know a dictionary $\mathbf{B}$, with basis $b_i, i = 1, 2, \cdots, n$ (each $b_i \in \mathbb{R}^d$) as its columns, in which v is k-sparse ($k \ll d$). Sparse coding refers to the idea of finding these k basis whose linear combination will approximate v. Contrary to classical approaches such as PCA, sparse coding allows $\mathbf{B}$ to be *overcomplete* (and thus non-orthonormal), making it more flexible to adapt to the data. Such representations have found immense success in computer vision: a few well-known examples include image compression using wavelet dictionaries ([39]), image denoising ([17]), and inpainting ([18]). Given v and $\mathbf{B}$, the problem of finding the k-sparse vector $w = S_k(v)$ can be cast as the following optimization problem:

$$\underset{w}{\operatorname{argmin}} \|v - \mathbf{B}w\|_2^2, \quad \text{subject to} \quad \|w\|_0 = k, \tag{1}$$

where the ℓ_0 pseudo-norm (which is the number of non-zeros in the coefficient vector w) is constrained at k. Unfortunately, this formulation results in a combinatorial optimization problem that is known to be NP-hard. As a result, relaxations to (1) have been proposed in which one replaces the ℓ_0 constraint with its closest convex counterpart, the ℓ_1 norm ([13]). This results in the following optimization problem (popularly known as the LASSO in the statistics literature ([47])):

$$\mathrm{SC}_{\text{LASSO}}(t; v) := \underset{w}{\operatorname{argmin}} \|v - \mathbf{B}w\|_2^2, \quad \text{subject to} \quad \|w\|_1 \leq t, \tag{2}$$

for a tuning paramter t. Typically, the formulation in (2) is written in the equivalent *Lagrangian* form (known as *Basis Pursuit Denoising* ([6])):

$$\text{SC}_{\text{BP}}(\lambda; v) := \underset{w}{\text{argmin}} \frac{1}{2}\|v - \mathbf{B}w\|_2^2 + \lambda\|w\|_1, \tag{3}$$

for a regularization parameter $\lambda > 0$ (which is inversely related to the bound t in (2)). Over the past few years, there have been several algorithms suggested to solve the LASSO efficiently ([38, 14, 30]).

A fundamental problem that one has to deal with when applying sparse coding to data from a new application is to design an *appropriate dictionary* in which the data points can be sparsely approximated. To address this difficulty, Dictionary Learning (DL) techniques have been suggested, in which one learns a dictionary from the data itself by solving a variant of (3), but with the dictionary also as an unknown. Using the notations in (3), and further assuming that we have access to a set of data vectors $v_i, i = 1, 2, \cdots, N$, the DL problem is stated as follows:

$$\min_{\mathbf{B}, w_1, w_2, \cdots, w_N} \sum_{i=1}^{N} \|v_i - \sum_j w_i^j b_j\|_2^2 + \lambda\|w_i\|_1 \quad \text{s.t.} \quad \|b_j\|_2 \leq 1, \ \forall j \in \{1, ..., n\}, \tag{4}$$

where the notation w_i^j denotes the jth dimension of the coefficient vector w_i. Further, to disallow arbitrarily large values in the learned basis, each basis is constrained to have unit norm. Although, the DL formulation (4) is non-convex and non-differentiable, there have been several approaches advocated to solve it to a local minimum efficiently, such as MOD of [19], K-SVD of [1], and online DL of [34].

A quantity that we frequently allude to in this paper is the *coherence* of the dictionary, which is defined as follows:

Definition 1 (Coherence)**.** *Assume that* $\mathbf{B}$ *is a dictionary with* $b_1, b_2, \cdots, b_n$ *as its columns, learned as per* (4)*. Then we define the* coherence μ *of* $\mathbf{B}$ *as:*

$$\mu(\mathbf{B}) = \max_{i<j} |b_i^T b_j|, \ \forall i, j \in \{1, 2, \cdots, n\}. \tag{5}$$

Another important ingredient of our algorithm is the *Regularization Path* (RP) of the sparse codes. Recall that, the greater the regularization on the ℓ_1 norm (in (2) or (3)), the sparser w will be. It can be shown that the coefficients in w corresponding to the active basis form piecewise linear trajectories ([36]) as a function of λ; RP refers to this path. Formally,

Definition 2 (Regularization Path)**.** *Referring to* (3)*, let* $w(\lambda) = \text{SC}_{BP}(\lambda; v)$*, and let* $w^i(\lambda)$ *be the i-th dimension of* $w(\lambda)$ *corresponding to a basis* $b_i \in \mathbf{B}$*. Then, we define the* Regularization Path *of* b_i *as the set of all values of* $w^i(\lambda), \forall \lambda \in [0, \infty)$*.*

3.2 Least Angle Regression (LARS)

Amongst several efficient algorithms for sparse coding that solves either the formulation (2) or (3) ([37, 30, 26, 20]), we will be particularly interested in the *Least Angle Regression* (LARS) algorithm of [14]; reasons for which will be described after reviewing this algorithm briefly below.

The LARS algorithm essentially solves the LASSO formulation in (2). It starts with a coefficient vector $w = 0$ and builds the sparse w in steps. At step k, let us assume θ^k is an estimate of v. Each step of LARS finds a basis from the dictionary $\mathbf{B}$ that is most correlated with the residual $C^k = v - \theta^k$ at that step. Let us say $b^* \in \mathbf{B}$ is such a basis at step k. Assuming $\mathbf{B}_\mathcal{A}$ is a set of active basis vectors corresponding to the support set $\mathcal{A}$ of w^k, then LARS will compute a descent direction u that is equiangular to b^* and all the basis in $\mathbf{B}_\mathcal{A}$. Next, LARS does steepest gradient descent in the direction u to produce a new estimate for v: $\theta^{k+1} = \theta^k + \gamma^k u$. The step size γ^k is chosen so that after this single descent, the residual C^{k+1} will be equally correlated with u and some other basis $b' \in \mathbf{B} \setminus \mathbf{B}_\mathcal{A}$. The efficiency of LARS comes from the fact that γ^k can be precomputed. This procedure is repeated until the ℓ_1 constraint on w is violated or the residual is lower than a pre-specified threshold. Algorithm 1 reviews the essential steps of LARS (refer to [14] for details).

LARS has several useful properties from the perspective of retrieving nearest neighbors, namely (i) the computation of equiangular directions naturally provides a radial partitioning of the data space; each new basis added to the active set $\mathcal{A}$ splits the space into partitions of smaller sizes, and (ii) starting with an empty active set, LARS offers an efficient way to compute the entire trajectories of the basis in the active set, in a single pass. Lower regularizations (or larger bounds t in Algorithm 1) lead to closer approximations to the data. These two properties help control the approximation to the ANN we desire. Further, LARS is *less* greedy compared to other sparse coding approaches such as *Orthogonal Matching Pursuit* ([38]); as a result improves the robustness of the sparse code to data perturbations, as we will see later.

Algorithm 1 LARS

Require: $\mathbf{B}$, v, t {or δ such that $\|v - \mathbf{B}w\|_2 \leq \delta$}

1: $k \leftarrow 0, \theta^k \leftarrow 0, \mathbf{B}_\mathcal{A} \leftarrow \{\}$
2: **while** $\|w^k\|_1 \leq t$ (or $\|v - \mathbf{B}w^k\|_2 > \delta$) **do**
3: $\quad C^k \leftarrow (v - \theta^k)$
4: $\quad b^* \leftarrow \text{argmax}_{\forall b \in \mathbf{B} \setminus \mathbf{B}_\mathcal{A}}(|b^T C^k|)$
5: $\quad \mathbf{B}_\mathcal{A} \leftarrow \mathbf{B}_\mathcal{A} \cup \{b^*\}$
$\quad$ {Let $\mathbf{1}_\mathcal{A}$ is a vector of ones with $|\mathcal{A}|$ dimensions, the *equiangular vector* u is given by:}
6: $\quad u \leftarrow \mathbf{B}_\mathcal{A} w^k_\mathcal{A}$, where $w^k_\mathcal{A} \leftarrow \left(\mathbf{1}'_\mathcal{A} \left(\mathbf{B}^T_\mathcal{A}\mathbf{B}_\mathcal{A}\right)^{-1} \mathbf{1}_\mathcal{A}\right)^{-\frac{1}{2}} \left(\mathbf{B}^T_\mathcal{A}\mathbf{B}_\mathcal{A}\right)^{-1} \mathbf{1}_\mathcal{A}$
7: $\quad \theta^{k+1} \leftarrow \theta^k + \gamma^k u$; Refer [14] for details on γ^k
8: $\quad k \leftarrow k + 1$
9: **end while**
10: **return** w^k

However, the overcompleteness of $\mathbf{B}$ poses an issue when using LARS. Recall that, the basis in $\mathbf{B}$ are linearly dependent. As a result, it is possible that the direction of LARS descent at some step might be opposite to the direction of some other basis already in the active set. This might result in the decrease or even zero-crossing of the coefficient associated with this basis in w. When this happens, LARS will drop this basis from the active set [14]. Such a basis drop will violate the data space partitioning assumptions that we alluded to earlier. Thus, we assume that LARS will not encounter any *basis drop*, explicit conditions for which will be provided in Section 5.3. Formally,

Condition 1. *For a data vector v, let $w^k = S_k(v)$ be the k-sparse vector found by LARS at step k and let w^k_i denote the i-th dimension of w^k. Then, if $w^k_i \neq 0$, implies $w^{k+1}_i \neq 0$.*

The following propositions directly follow from Algorithm 1 and Condition 1.

Proposition 1. *With reference to the LASSO formulation* (2) *and assuming Condition 1, let v be k-sparse in $\mathbf{B}$ and assume that w^j be the sparse vector found by LARS at the j-th step ($0 < j \leq k-1$). Then,*

1 *The cardinality of the LARS support increases in each LARS step, i.e. $\|w^j\|_0 = j$ and $\|w^{j+1}\|_0 > \|w^j\|_0$.*

2 *Increasing the bound decreases sparsity, i.e. there exist bounds $t, t', t < t'$, so that $\|\text{SC}_{LASSO}(t; v)\|_0 = j$ and $\|\text{SC}_{LASSO}(t'; v)\|_0 = j + 1$.*

Proof. Considering each case,

1 This follows directly from Algorithm 1, if Condition 1 is imposed.

2 Let $\|C^j\|_2$ be the minimizer of (2) for a bound t and assuming Condition 1. Given that v is k-sparse and $j \leq k-1$, there must exist a residual C such that $\|C\|_2 < \|C^j\|_2$. Given the convexity of the formulation and the steepest gradient descent at each LARS step, we can achieve $\|C\|_2$ only by continuing the loop in Algorithm 1. This implies that we need to increase the bound to $s > t$. Now, given that LASSO regularization path is continuous and piecewise linear (refer [36][Lemma 2]), there should exist a $t < t' \leq s$ such that $\|\text{SC}_{\text{LASSO}}(v; t')\|_0 = j + 1$. □

When the Coherence $\mu(\mathbf{B}) = 1$, there will be multiple equivalent LARS regularization paths (which deters its usefulness to ANN retrieval as we cannot produce unique sparse codes for a given data vector). The following proposition formally states this fact.

Proposition 2. *Assuming Condition 1, for a data vector v, let $\mathcal{A} = \{b_{i1}, b_{i2}, \cdots, b_{ik}\}$ be a well-ordered set of basis vectors $b_{ij} \in \mathbf{B}$ selected by LARS, with the ordering $b_s < b_t$, if $s < t$ for LARS steps s and t. Then the sequence $\mathcal{A}$ is* unique *iff $\mu(\mathbf{B}) < 1$.*

Proof. It is easy to see that if $\mu(\mathbf{B}) = 1$, step 4 of Algorithm 1 will select multiple equally correlated bases. □

Now, we have all the necessary results to establish a connection between NN retrieval and sparse coding.

4 Sparse Coding and ANN Retrieval

To make our proofs more intuitive, we will use the *Constrained BP* (CBP) form of the LASSO formulation ([52]), which constrains the reconstruction error as opposed to sparsity:

$$\mathrm{SC}_{\mathrm{CBP}}(\delta; v) := \underset{w}{\operatorname{argmin}} \|w\|_1 \quad \text{subject to} \quad \|v - \mathbf{B}w\|_2^2 \leq \delta^2. \tag{6}$$

4.1 ℓ_2 Distances in Sparse Space

The following result establishes a connection between the ℓ_2 norm distances between the data vectors in the input space and their sparse codes.

Theorem 1 (Distances)**.** *Let $v_1, v_2 \in \mathbb{R}^d$ be two zero-mean data points and let $\mathbf{B}$ a given dictionary. Further, let $w_1 = S_k(v_1)$ and $w_2 = S_k(v_2)$ be the k-sparse vectors respectively, produced as per the formulation SC_{CBP} given in (6) so that $\|v_i - \mathbf{B}w_i\|_2^2 \leq \delta^2$, for $i \in \{1, 2\}$. Then, we have:*

$$\|v_1 - v_2\|_2 \leq 2\delta + \|\mathbf{B}\|_2 \|w_1 - w_2\|_2. \tag{7}$$

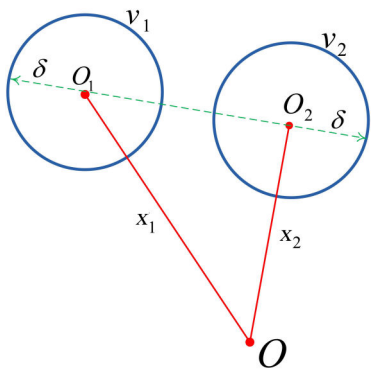


Figure 1: With reference to Theorem 1: O is the origin, $x_i = \mathbf{B}w_i, i \in \{1, 2\}$ are two approximate reconstructions of the two zero-mean data vectors v_1 and v_2 respectively. The hyperspheres each of radii δ and centers O_1 and O_2 represent the locus of the two data points given their approximate reconstructions.

Proof. Let us assume that $x_1 = \mathbf{B}w_1$ and $x_2 = \mathbf{B}w_2$. With reference to Figure 1, considering the triangle formed by O, O_i and points v_i on the hypersphere with center O_i (for $i \in \{1, 2\}$) respectively, we have

$$\|v_1 - v_2\|_2 \leq \max_{r_1, r_2} \|x_1 + r_1\delta - x_2 - r_2\delta\|_2, \tag{8}$$

where δ is the radius of the ℓ_2 balls shown in Figure 1 and r_1, r_2 are unit norm directions. After a few rearrangements in the RHS, we get to the result. □

Typically, $\|B\|_2$ is around 10 for all the datasets that we use in our experiments. Building on Theorem 1, we will now introduce our data representation for ANN retrieval.

5 Subspace Combination Tuple

As is clear from our previous discussions, each basis in the dictionary represents a non-orthogonal subspace. Through sparse coding we are in fact seeking a new subspace that is a sparse linear combination of the subspaces defined by these basis, that will host a given data vector. If the data point is exactly k-sparse in $\mathbf{B}$, then there is a unique combination of k subspaces

from the dictionary for this data vector.[1] In other words, the k basis can be used as *representatives* of a subset of the data, enabling data partitioning. The main idea of this paper is to build a hashing algorithm over these representative basis. First, let us define an encoding of the active support of a k-sparse data vector.

Definition 3 (Subspace Combination Tuple (SCT))**.** *Suppose that $w \in \mathbb{R}^n$ is k-sparse. Further, let $\mathcal{I} = \{1, 2, \cdots, n\}$ be the set of the first n natural numbers. We define the mapping $h_k : \mathbb{R}^n \to I$, ($I \subset \mathcal{I}$, $|I| = k$) as a* Subspace Combination Tuple *of w, if $h(w) = \{i_1, i_2, \cdots, i_k\}$, where i_j represents the index of the dimensions of w, and $i_j \in h(w)$ if $w_{i_j} \neq 0$, $\forall i_j \in \mathcal{I}$.*

To make our representation clear, let us take a simple example.

Example 1. *Assume that for a data vector $v \in \mathbb{R}^{100}$, we have $w = S_3(v)$ in some $\mathbf{B}$, where $w \in \mathbb{R}^{1000}$. Let us further assume that* $\text{Supp}(w) = \{w_{25}, w_{49}, w_{972}\}$ *correspond to the non-zero dimensions in w. Then, we have $h_3(w) = \{25, 49, 972\}$ (the exact ordering of these numbers is irrelevant at this point).*

The following theorem establishes a connection between the SCTs and the angle between the corresponding data vectors.

Theorem 2 (Angles)**.** *For any two zero-mean unit-norm data vectors v_1 and v_2, let $w_i = \text{SC}_{CBP}(\delta; v_i)$ and $T_i = h_k(w_i)$ for $i \in \{1, 2\}$ be the respective k dimensional SCTs produced by LARS using a dictionary $\mathbf{B}$ such that $\mu(\mathbf{B}) < 1$. Suppose $\|v_1 - v_2\|_2 \leq 2\delta$, then for a probability density defined on a Lebesgue measure given by the volume of overlap of the two hypershperes with centroids v_i, loci $x_i = \mathbf{B}w_i$ and radii δ, Prob $(|T_1 \cap T_2| = k)$* monotonically increases *with decreasing $\widehat{v_1, v_2}$.*

Proof. Since v_i are zero-mean unit norm, we have $\widehat{v_1, v_2} \propto \|v_1 - v_2\|_2$. As the centroids v_i and the radii δ of the two hyperspheres are fixed, they must have a non-zero intersection when $\|v_1 - v_2\|_2 \leq 2\delta$. Using the fact that the volume of intersection of two hyperspheres increases monotonically with decreasing centroid distances (See [21][Theorem 1.A]), we have $Prob\,(|T_1 \cap T_2| = k) \propto |T_1 \cap T_2|/|T_1 \cup T_2| \propto 1/\|v_1 - v_2\|_2 \propto 1/\widehat{v_1, v_2}$. Now, assuming Proposition 2, the result follows (the restriction $\mu(\mathbf{B}) < 1$ and the convexity of SC_{CBP} confirms that no basis in $\mathbf{B}$ that may produce different w_is for the same minimizing objective). □

Theorem 2 can be used to design a hashing algorithm on the SCT representation for ANN retrieval. In fact, such a scheme relates to the well-known Min Hash algorithm ([5]). Conversely, the SCT representation cannot weigh each active coefficient according to its importance. This is critical especially when some of the coefficients may have small values such that they are vulnerable to data perturbations. To avoid this issue, we will describe a modified version of SCT, which we will call an *Ordered Subspace Tuple*. We will show later that OST helps us to generate robust hash codes, together with providing theoretical guarantees for our algorithm. It will also help us circumvent several important issues that we might need to face when using our scheme in practice, as will be explained later. We define OST as follows:

Definition 4 (Ordered Subspace Tuple: OST)**.** *Given a data vector v and a dictionary $\mathbf{B}$. Suppose $w = \text{SC}_{CBP}(\delta; v)$ be the k-sparse code produced by LARS. Then, an SCT $h_k(w) = \{i_1, i_2, \cdots, i_k\}$ is defined as an Ordered Subspace Tuple (OST), if $i_1, i_2, \cdots, i_k$ are arranged in the order in which the respective bases b_{i_j} are* activated *by the LARS algorithm (where b_{i_j} corresponds to the i_jth column of the dictionary $\mathbf{B}$). We will denote a k dimensional OST associated with a data vector v as $\overset{\circ}{h}_k(v)$ (the accent $\circ$ denoting 'order') and will be synonymous with a* hash code*.*

Theorem 2 connects the *nearness* of two data vectors and the angles between them, which is further related to matching their OSTs. This idea helps to create hash codes on the OSTs, so that data vectors that make closer angles will be mapped to the same hash bucket. This scheme by itself will not help retrieving nearest points, since two data vectors can be close in angles, but large in radial distances. Fortunately, when combined with Theorem 1, this issue can be avoided. Algorithm 2 describes our OST-Hashing scheme in detail.

5.1 Data/Space Partitioning

Considering that we have introduced our hashing scheme in detail, Figure 2 illustrates the partitioning introduced by the basis as the support size increases. Since the data and thus the dictionary are assumed to be zero-mean, all the basis are shown to pass through the origin. The red points show 2D data points for which a dictionary of size 2×7 is learned. A 1-sparse reconstruction is essentially the basis themselves, while higher support sizes introduce an increasing number of basis combinations (in equiangular directions). An OST represents one of such lines and all the data points angularly proximal

[1] Here we assume that we use the LARS algorithm, which provides a unique regularization path as described in Proposition 2

Algorithm 2 OST-Hashing

Require: $\mathbf{B}$, $\mathcal{D}$, δ, H\{a hash table indexed by OST\}

1: **for all** $v_i \in \mathcal{D}$ **do**

2: compute $w_i = \text{SC}_{\text{CBP}}(\delta; v_i)$ and $T_i = \overset{\circ}{h}_k(v_i)$.

3: $H(T_i) \leftarrow H(T_i) \cup w_i$ {assign a location to w_i in the hash bucket $H(T_i)$}

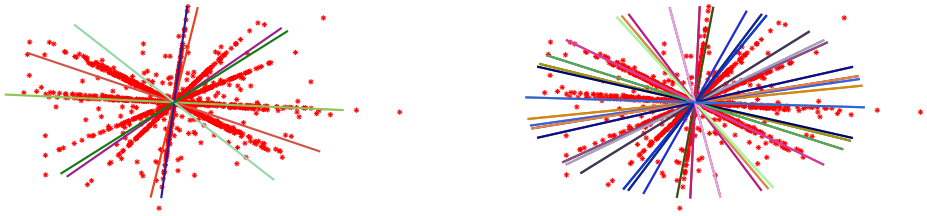
4: **end for**

5: Given a query q, compute $T_q = \overset{\circ}{h}_k(q)$, and find the ANN of q

$$w^* = \underset{\forall w \in H(T_q)}{\text{argmin}} \|w - w_q\|_2 \tag{9}$$

where $w_q = \text{SC}_{\text{CBP}}(\delta; q)$.

6: **return** $v^* \in \mathcal{D}$ associated with w^*.



(a) 1-sparse (b) 2-sparse

Figure 2: Data space partitioning induced by sparse coding. The red points represent 2D data points and the lines denote the dictionary bases from a 2×7 learned dictionary. 2(a) shows the directions of the dictionary bases (corresponding to a 1-sparse representation), 2(b) shows equiangular directions corresponding to all possible 2 bases combinations from the dictionary (2-sparse).

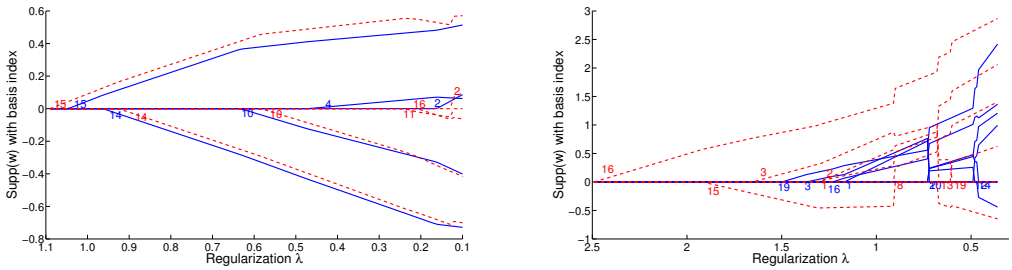
to a line are mapped to a unique hash bucket associated with that line. Points may vary in radial distances; in which case we use a linear scan or a k-d tree to find the ANN from these radial subsets. Note that since we need to store *only* the coefficients of the sparse codes associated with the data points, our storage needs are minimal. More to it, our search in a hash bucket is over low dimensional spaces (corresponding to k-sparse codes); as a result we do not need to worry about the curse of dimensionality of the original high-dimensional data any more.

5.2 Regularization Path and OSTs

In Figure 3, we illustrate how the regularization paths of two data points evolve as a function of the regularization value λ. This illustration has two motives, namely (i) it helps provide an intuition into how basis selection varies for two nearby and two far away (in Euclidean distance) data points, and (ii) it provides an intuition into a scheme which we will introduce in the next section for generating robust hash codes. The two data points v_1 and v_2 (each $v_i \in \mathbb{R}^{10}$) for the experiment are generated from the same dictionary $\mathbf{B} \in \mathbb{R}^{10 \times 20}$ and same underlying sparse vector w, but added with different amounts of Gaussian noise. In Figure 3(a), we use $\|v_1 - v_2\|_2 = 0.2$, while for Figure 3(b), $\|v_1 - v_2\|_2$ is 1.5. The figures plot the non-zero dimensions in w (along with indicating the corresponding non-zero index). From the below plots we learn that for small Euclidean distances, the activation of the two data points for various regularizations λ follow the order closely (that is, $\overset{\circ}{h}_3(v_1) = \overset{\circ}{h}_3(v_2) = \{15, 14, 10\}$), while for large Euclidean distances, the OSTs are completely different. We would also like to point out from Figure 3(a) that, increasing regularizations improve the overlap between the two OSTs.

5.3 Length of Hash Codes

Determining the appropriate length of our hash code is of atmost importance affecting both the speed of sparse coding, as well as the accuracy of retrieval. Smaller OSTs lead to too many hash table collisions, while longer hash codes lead to smaller data partitions and thus more sensitive OSTs. Further, recall the issue of dropping basis brought about by the LARS algorithm



(a) $\|v_1 - v_2\|_2 = 0.2$ (b) $\|v_1 - v_2\|_2 = 1.5$

Figure 3: Illustrations showing the connections between the Regularization Paths (RP), Euclidean distances, and overlapping hash codes of two data points. Left plot shows the RPs for two data points close in the data space and right plot shows the RPs for two far away data points. The colors blue and red indicate the paths for the two data points against increasing regularization λ. We also show the index of the corresponding non-zero dimensions for each data point whenever it is activated along the path.

(described in 3.2). This issue suggests that we should select the length of our hash code (that is, number of elements in the OST) in such a way that any combination of the corresponding basis will not cause any basis drop. Such an idea is possible only if we can ensure that the basis in the active set form a linearly independent set, bringing in the notion of the *Spark* of a given dictionary.

Definition 5 (Spark ([15]))**.** *Spark of a dictionary* $\mathbf{B}$ *is defined as the smallest number of columns from* $\mathbf{B}$ *that are linearly dependent. That is,*

$$\text{Spark}(\mathbf{B}) = \min_{\mathbf{B}x=0} \|x\|_0. \tag{10}$$

Using Spark, we can formally state our condition for *no basis drop* as follows:

Theorem 3. *For a data vector* v*, suppose* $T = \mathring{h}_k(v)$ *is the* k *dimensional OST generated using a dictionary* $\mathbf{B}$ *via the LARS algorithm. Then, there will not be any basis drop if* $k < \text{Spark}(\mathbf{B})$*.*

Although this idea is appealing, computing Spark is NP-hard, and thus we need to seek approximations to it. One useful result along these lines is a lower bound on the Spark using dictionary coherence, stated as follows:

Theorem 4. *Suppose* $\mu(\mathbf{B})$ *is the coherence of a dictionary* $\mathbf{B}$*, then we have:*

$$\text{Spark}(\mathbf{B}) \geq 1 + \frac{1}{\mu(\mathbf{B})}. \tag{11}$$

Proof. See [15][Lemma 2.1] for the proof. □

This lower bound can be too restrictive when coherence is high, which is often the case in practice. Thus, instead of using the it directly, we will use the average coherence in Theorem 4 to decide the OST length. We define the *mean coherence* as follows:

Definition 6. *For a dictionary* $\mathbf{B}$ *with* n *basis, we define the* mean coherence *(*μ_m*) as:*

$$\mu_m(\mathbf{B}) = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} |b_i^T b_j|, \ \forall b_i \in \mathbf{B}. \tag{12}$$

Although, it might seem we need to sacrifice the accuracy of NN retrieval using this scheme, this performance drop can be easily compensated by using hash codes of different lengths as suggested in the last section. This is further substantiated empirically in our experiments.

6 Multi-Regularization Sparse Coding

In this section, we introduce our main algorithm to produce robust hash codes for efficient ANN retrieval. Recall that DL fits multiple non-orthogonal subspaces to the dense regions of the data, leading to smaller partitions for increasing support sizes. On the positive side, this leads to the retrieval of ANN that are most similar to the query points. On the negative side, this scheme can be too demanding. Most of the real world data descriptors are prone to data perturbations in the form of noise or image distortions. When the data partitions are small, even the slightest perturbation can get the data point to change the partition, leading to mismatching OSTs. Another problem is that of k-NN retrieval, (for $k > 1$). Small partitions might not have enough data points to guarantee a k-NN retrieval. To address these concerns, we will advocate a hierarchical sparse coding algorithm in this section that utilizes the regularization path of LARS to generate robust OSTs. Before we discuss our scheme, let us recall a few basic facts about the LARS algorithm.

Lemma 1. *Assuming a dictionary* $\mathbf{B}$*, such that* $\mu(\mathbf{B}) < 1$*. For a data vector* v*, if* $b_i \in \mathbf{B}$ *is the most correlated basis to* v*, then* b_i *will be selected in the first step of LARS.*

Lemma 2. *Let* $w = \mathrm{SC}_{CBP}(\delta; v)$ *be the sparse code associated with a data vector* v *using a dictionary* $\mathbf{B}$ *and reconstruction error bound* δ*. Then* $w = 0$*, if* $\delta \geq \|v\|_2$*.*

Proof. Recall that for LARS steps k and $k+1$, we have the following relation with regard to the reconstruction errors (see Proposition 1).

$$\|v - \mathbf{B}w^{k+1}\|_2 < \|v - \mathbf{B}w^{k}\|_2. \tag{13}$$

Now, we know that LARS continues its loop until the error bound $\|v - \mathbf{B}w\|_2 \leq \delta$ is satisfied. From (13), it is clear that every step of LARS will generate smaller errors. So the maximum error that is possible is when $w = 0$, which corresponds to $\|v\|_2$. Thus, for any $\delta > \|v\|_2$, the error bound condition will be satisfied and LARS will produce $w = 0$. □

In Proposition 1, we showed that by adjusting the LASSO regularization (or the bound t or δ), we can control the sparsity of the solution, i.e. the length of the OST hash code. The next theorem connects the regularization path of the LASSO solution produced by LARS, with matching hash codes.

Theorem 5. *Assume* v_1*,* v_2 *are two zero-mean unit norm data vectors, and* $\mathbf{B}$ *is a given dictionary with coherence* μ*. Let* $w_i = \mathrm{SC}_{CBP}(\delta; v_i)$ *(for* $i \in \{1, 2\}$*) be the sparse code associated with each of the data points respectively for a reconstruction error bound* δ*. Suppose* $T_i = \mathring{h}(w_i)$ *be the respective OSTs. Suppose* $T_1 \neq T_2$*, then there exists a* δ'*, where* $\delta < \delta' < 1$ *so that for* $w_i' = \mathrm{SC}_{CBP}(\delta'; v_i)$ *and the associated OSTs* $T_i' = \mathring{h}(w_i')$*, we have:*

I. $T_1' \subseteq T_1$ *and* $T_2' \subseteq T_2$

II. $T_1' = T_2'$

iff there exists a $b_j \in \mathbf{B}$ *such that* $|v_i^T b_j| < \sqrt{\frac{1}{2}(1+\mu)}$*, for* $i \in \{1, 2\}$*.*

Proof. Considering each case:

I. This is a direct result from the continuity properties of the LARS solutions as implied by Proposition 1, when Condition 1 is satisfied.

II. Let us define a θ such that $cos\theta = \mu$ so that $\sqrt{\frac{1}{2}(1+\mu)} = cos\frac{\theta}{2}$. To prove the *if* part: suppose $\exists b_i \in \mathbf{B}$ such that $|b_i^T v| < \sqrt{\frac{1}{2}(1+\mu)}$, then $\widehat{b_i, v} < \frac{\theta}{2}$. This condition means that b_i is the most correlated dictionary basis to v and thus will be selected by LARS in the first step. If not, there must exist another basis b_j such that this condition is satisfied. In such a case, $b_i^T b_j < \mu$, which contradicts the condition that the coherence of the dictionary is μ. Now, using the definition of OST, combined with Proposition 1 and Lemma 2, we have that $\exists \delta'$ such that $\delta < \delta' < max(\|v_1\|_2, \|v_2\|_2) = 1$(data vectors are assumed unit norm), which satisfies $T_1 = T_2$. To prove the *only if* part: assume that there exists a $0 \leq \delta < max\left(\|v_1\|_2, \|v_2\|_2\right)$ so that $T_1 = T_2$. Let $T_1 = T_2 = \{i1, i2, \cdots, ik\}$ be the respective OST. Then by the definition of OST, we have that $i1$ is the most correlated basis to the data v and thus using Lemma 1 will be selected at the first step of LARS, from which the result follows.

□

We have from Theorem 5 that, if the OST representations for two data vectors that make an angle less than $\theta/2$ with the same basis do not coincide at one error bound, then there should exist a larger bound at which the two OSTs will accord. This implies that if we use multiple OST codes corresponding to a set of bounds, then at least one of them will match up with a query OST (provided the required conditions stipulated in Theorem 5 are satisfied). This is exactly the idea in the Multi-Regularization Sparse Coding (MRSC) algorithm we propose below. There is one key difference though; instead of running the LARS algorithm in multiple passes for each bound, we make a single pass, keeping track of the index of active basis support at each LARS iteration, incrementally building the OST. This is theoretically equivalent to running the algorithm for different bounds. Using this idea, we can create multiple OST for a sequence of support sizes: $\mathcal{L} := \{L_{min}, L_{min}+1, L_{min}+2 \cdots, L_{max}\}$. Thus, we define an *OST hash code set* $H_{\mathcal{L}}(v)$ as follows:

$$H_{\mathcal{L}}(v) := \{\mathring{h}_{L_{min}}(v), \mathring{h}_{L_{min}+1}(v), \cdots, \mathring{h}_{L_{max}}(v)\}. \tag{14}$$

Algorithms 3 and 4 show the essential steps of our MRSC scheme. [2]

Algorithm 3 MRSC Hashing

Require: $\mathbf{B}$, $\mathcal{D}$,
1: Initialize hash table $\mathcal{H}$
2: **for** $v_i \in \mathcal{D}$ **do**
3: **for** $\ell \leftarrow \{L_{min}, \cdots, L_{max}\}$ **do**
4: $w \leftarrow \mathrm{SC}_{\mathrm{CBP}}(\delta_\ell; v_i)\{\delta_\ell$ is an error bound corresponding to an ℓ-sparse OST.$\}$
5: $T \leftarrow \mathring{h}_\ell(v_i)$
6: $\mathcal{H}(T) \leftarrow \mathcal{H}(T) \cup w\{$ assign w to a hash bucket associated with $T\}$
7: **end for**
8: **end for**
9: **return** $\mathcal{H}$

Algorithm 4 MRSC Query

Require: $\mathbf{B}$, $\mathcal{H}$, v_q, $\mathrm{NN}_q = \{\}$
1: Initialize $\mathrm{NN}_q \Leftarrow 0$
2: **for** $\ell \leftarrow \{L_{max}, \cdots, L_{min}\}$ **do**
3: $w_q \leftarrow \mathrm{SC}_{\mathrm{CBP}}(\delta_\ell; v_q)\{\delta_\ell$ is an error bound corresponding to an ℓ-sparse OST.$\}$
4: $T \leftarrow \mathring{h}_\ell(v_q)$
5: **if** $\mathcal{H}(T) \neq \{\}$ **then**
6: $w^* \leftarrow \mathrm{argmin}_{w \in \mathcal{H}(T)} \|w - w_q\|_2$
7: $\mathrm{NN}_q \leftarrow v^*$ associated with w^*
8: **return** NN_q
9: **end if**
10: **end for**
11: **return** NN_q

6.1 Improving Robustness

Let v be a unit-norm zero-mean data point and let $w = S_k(v)$ so that $\mathcal{S} = h_k(w)$ is an SCT associated with v. Suppose ϵ is some noise such that we have $v' = v + \epsilon$. Suppose $\mathcal{S}' = h_{k'}(v')$ be the respective SCT. Then, as long as $\|\epsilon\|_2 < 2\delta$ (for a reconstruction error bound δ), $Prob\,(\mathcal{S} \cap \mathcal{S}' \neq \{\}) > 0$ (as we saw in Theorem 2). Note that our MRSC algorithm starts with a primary data partition produced by the basis most correlated to the data point. Under perturbations, this primary basis might change, but still maintains the distance 2δ.

To make our hashing to be robust to such descriptor variations, we propose the following heuristic. Instead of generating a single MRSC, we do a cyclic rotation of the MRSC hash codes, making each of the basis indices as the first entry in each

[2]Note that our scheme is generic and does not assume LARS specifically. In case we use a sparse coding algorithm that does not have a regularization path (such as [20]), still we can use our algorithms by explicitly specifying the error bounds δ_ℓ. Typically, δ_ℓ can be varied in fixed steps.

cycle. This results in our algorithm to query subspaces majored by other indices in the hash code. If we assume that the length of the OSTs are short, so that computing all permutations of the OST is not costly, then we can in fact look at every permutation (each permutation is an OST). This provides faster search of the subspaces. Further, such a heuristic can be easily implemented in parallel. After retrieving the neighbors from each of the OSTs in the cycle (or permutations), the ANNs are decided by sorting the candidate neighbors with respect to the distances between the query and the neighbors' sparse codes. Referring back to Algorithm 4, such a heuristic will have an additional loop at line#5, where the T is rotated (or permuted).

6.1.1 Hierarchical Sparse Coding

It is possible that the given data vectors have minor correlation such that there is no overlap between their dictionary support. A way to tackle this issue through our framework is to use *hierarchical sparse coding* in which we learn multiple dictionaries each with number of basis vectors reduced by half. That is, instead of sparse coding the data point $v \in \mathbb{R}^n$ with a dictionary with m bases, we generate k different sparse codes for dictionaries of sizes $\{m, \frac{m}{2}, \frac{m}{4}, \cdots, \frac{m}{2^k}\}$; $k \leq n$. Intuitively, the coherence of the dictionaries (and thereby the subspaces covered) decrease as the dictionary size decreases, such that each dictionary covers a larger subspace, addressing the issue of getting the nearest neighbor. This is a topic to be analyzed on its own and we consider it as part of a future extension to this paper.

7 Algorithm Analysis

Computational Complexity: Let us assume that we use a maximum of k iterations of the LARS algorithm for sparse coding a data point $v \in \mathbb{R}^d$ using a dictionary $\mathbf{B} \in \mathbb{R}^{d \times n}$. Further, assuming we use Cholesky factorization and updation for computing the least angle direction in each iteration of the LARS algorithm; then each direction computation takes $\mathcal{O}(j^2)$ for the jth LARS step, leading to $\mathcal{O}(k^3)$ time for k steps. To compute the maximum correlated dictionary atom, for k steps we require $nd + (n-1)d + (n-2)d + ... + (n-k)d) = \mathcal{O}(ndk)$ computations. So total computational complexity is $\mathcal{O}(k^3 + ndk)$. As smaller codes ($k \leq 5$) are sufficient for our algorithm, the computation is dominated by the component $\mathcal{O}(ndk)$, which is the cost of making k inner products with the entire $\mathbf{B}$.

Space complexity: Applying Theorem 1, we need to store only the coefficients corresponding to the support in the SCT for computing the distances. Thus the space required is k memory locations for storing the coefficients and $k \log(n)$ bits for storing the SCT codes. Technically, as the active coefficients are stored as 4 byte floating point numbers, the total space is $\mathcal{O}(32k + k \log(n))$ bits. For example, using a support size of 5 and 2048 basis, we need 215 bits per data point of which 55 bits are used for hashing.

Query Complexity: Assuming the data $\mathcal{D}$ is uniformly distributed into the hash buckets by the dictionary $\mathbf{B} \in \mathbb{R}^{d \times n}$ Thus, for an SCT of size k, we have each hash bucket to contain $s = \frac{|\mathcal{D}|}{\binom{n}{k}}$ items. Assuming, we use linear search for resolving hash conflicts, a query takes $\mathcal{O}(s)$ time. If hash bucket corresponding to an SCT of size k is empty, then we need to retreat to an SCT of size $k-1$ and so on; the extra time for linear search that we need to do can be derived as follows: Let s_j denotes the time for doing linear search for an SCT of size j, then $\frac{s_{k-1}}{s_k} = \binom{n}{k} / \binom{n}{(k-1)}$ which leads to a complexity of $s_{k-1} = \mathcal{O}(k(n-k)s_k)$.

8 Experiments

In this section, we detail our experimental setup; the datasets used and the performance metrics against which the methods are evaluated. Our algorithms were implemented in C++ for speed, with MATLAB interfaces for ease of data handling. All the algorithms, except the one on speed of retrieval for an increasing database size, used a 2GHz 64bit machine with 4GB RAM.

8.1 Datasets

SIFT Dataset: Our experiments were primarily based on the SIFT descriptors created from the INRIA Holidays and the INRIA Copydays datasets[3]. We used approximately 400M SIFT descriptors created from 200K images from these datasets.

Spin Image Dataset: 3D shape recognition and retrieval has recently been gaining a lot of attention due to the introduction of cheap 3D sensors. Since these sensors produce millions of 3D points per video frame, retrieving corresponding points across frames for object recognition is a challenging operation. One of the most popular 3D point cloud descriptors is the *spin images* [25]. Each spin image is a 2D histogram of the changes in the 3D surface structure of the point cloud, computed in cylindrical coordinates, around a given point. We used a 20×20 histogram, thus forming spin vectors of 400D for each 3D point in the cloud. The descriptors were generated from the public SHREC dataset[4] [4], consisting of 50 object classes, of which we used 10 classes for computing the spin images; amounting to a total of 1M descriptors.

8.2 Performance Metrics

This subsection details the metrics against which we analyze the performance of the sparse coding algorithms.

Recall@K: Given a dataset $\mathcal{D}$ and a query set $\mathcal{Q}$, we define:

$$Recall@K = \frac{1}{|\mathcal{Q}|} \sum_{\forall q \in \mathcal{Q}} \mathrm{NN}_K^{\mathcal{A}}(q, \mathcal{D}) \cap \mathrm{NN}_1^{gt}(q, \mathcal{D}) \tag{15}$$

where $\mathrm{NN}_K^{\mathcal{A}}$ retrieves the K nearest neighbors of a query point q from $\mathcal{D}$ using the respective algorithm $\mathcal{A}$ and $\mathrm{NN}_1^{gt}(q, \mathcal{D})$ is the ground truth nearest neighbor computed using linear scan. This definition of recall has been used in many papers such as [24] as it provides a sense of approximately how many NNs have to be retrieved by a given algorithm so that there is a good chance that the true NN is retrieved.

Precision@K: Given a dataset $\mathcal{D}$ and a query set $\mathcal{Q}$, we define:

$$Precision@K = \frac{1}{|\mathcal{Q}|} \sum_{\forall q \in \mathcal{Q}} \mathrm{NN}_1^{\mathcal{A}}(q, \mathcal{D}) \cap \mathrm{NN}_K^{gt}(q, \mathcal{D}) \tag{16}$$

where NN_K^{gt} retrieves the K nearest neighbors of a query point q from $\mathcal{D}$ using linear scan and $\mathrm{NN}_1^{\mathcal{A}}$ is the nearest neighbor found by the algorithm $\mathcal{A}$. This definition of precision provides an intuition on the quality of the NNs retrieved by the respective algorithm; essentially if the dataset point as returned by the algorithm $\mathcal{A}$ belongs to the first K ground truth nearest neighbors.

8.3 Dictionary Learning

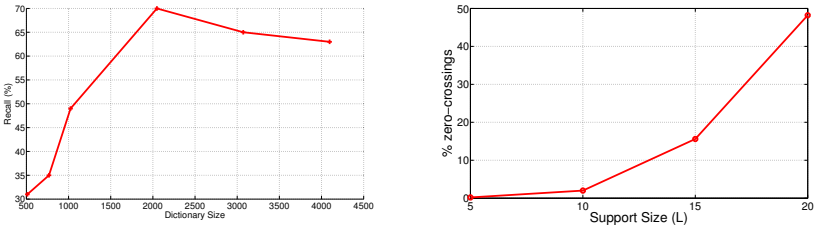
Learning the overcomplete basis dictionary is the first step in sparse coding; deciding the number of bases a critical component with regard to the accuracy and speed of sparse coding and NN retrieval. A dictionary with a large number of bases could increase the coherence between the atoms such that the sparse codes become too sensitive to noise or can even become ambiguous (coherence when unity). Too few dictionary atoms might result in less expressive SCTs. Thus, in this paper we approach the problem from a data driven perspective and use cross-validation against recall@1 to decide the optimum dictionary size.

Towards this end, we randomly chose 1M SIFT descriptors from the dataset. The dictionary learning was performed using the SPAMS toolbox [34, 33] for an increasing number of bases. Typically, SIFT descriptors have integer dimensions that pose significant problems with the standard dictionary learning algorithms, especially the ill-conditioning of the matrices involved. Thus we scaled down all the descriptors by dividing them by 255, and later normalizing each descriptor to have zero mean. Figure 4(a) shows the recall@1 performance on NN over SIFT descriptors for varying dictionary sizes on a small database of 10K descriptors and a query size of 1K.

[3] http://lear.inrialpes.fr/ jegou/data.php

[4] http://www.itl.nist.gov/iad/vug/sharp/contest/2011

As is clear from the plot, the larger the size of the dictionary, more time is taken for solving the LASSO. Lower number of bases lead to too many activations, leading to many hash collisions, thus reducing the performance. The best recall was observed for a dictionary of size 2048 for the SIFT descriptors and thus we use this dictionary for subsequent experiments in this paper. Figure 5 shows a typical 256 basis SIFT dictionary. Sparse coding a SIFT descriptor using a 2048 bases dictionary took approximately 100μs on an average. We fixed the LASSO regularization to 0.5 for dictionary learning. As for the spin image dataset, we followed suit and learned a dictionary of size 400×1600 using 100K descriptors from the dataset.



(a) DL (b) Basis Drop

Figure 4: (a) plot of the average recall performance of SIFT descriptors for varying sizes of the dictionary. The dataset was of size 10K SIFT descriptors and the query had 1K descriptors, (b) plot of the percentage of 100K SIFT descriptors that had at least one zero-crossing in its sparse regularization path against the allowed support set size.

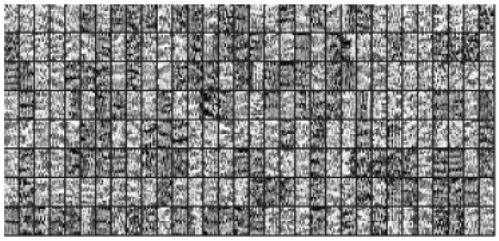


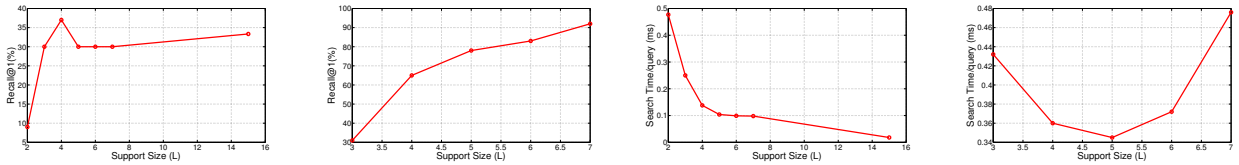
Figure 5: A sample SIFT dictionary with 256 bases learned from 1M training SIFT descriptors. Each basis is reshaped as a 16×8 patch in the display above.

8.4 Active Support Size

An important factor that needs to be estimated in our algorithm is the size of the active support. Smaller hash codes lead to too many collisions, while longer hash codes lead to slow sparse coding. As we described in Section 5.3, we decided to use the average coherence of the dictionary to determine the length of our hash codes. From the dictionaries we learned in the above step, the average coherence was found to be approximately 0.2, which suggests to use an OST length of 5. Since, avoiding basis drop is of at most interest to the correct working of our algorithm, we decided to substantiate this theoretical guarantee empirically on a subset of our SIFT dataset consisting of approximately 100K descriptors.

We computed the regularization paths of these descriptors by increasing the support set size from 1 to L, for $L = \{5, 10, 15, 20\}$. For example, for L=5, we computed LARS 5 times on a SIFT descriptor, each time incrementing the allowed number of supports by one (L is also the number of iterations of the LARS algorithm); the sign of each of the active basis in the set was then checked to see if any of the coefficients crossed the axis. Figure 4(b) shows the result of this experiment. The x-axis is the active set size (L) and y-axis plots the percentage of the dataset that had a zero-crossing at least once. We used a dictionary of size 2048 learned from 1M SIFT descriptors for this experiment. As is clear from the plot, the number of zero-crossings increase as we allow the activated basis atoms to live longer in the LARS process. For $L = 5$, we observed that less than 0.2% of the dataset produced zero-crossings, conforming to our theoretical choice. This number increased to approximately 50% for $L = 20$.

The next task is to determine the *range* of the regularizations (that is, the L_{min} and L_{max}) for our MRSC algorithm. Again, we estimate this range empirically on the SIFT descriptor dataset (as this is the dataset on which most of our experiments are based). To estimate the recall performance of NN against various support sizes, we did the following cross-validation experiment. On a subset of our SIFT dataset consisting of 1M SIFT descriptors, we computed the average recall@1



(a) OST (b) MRSC (c) OST (d) MRSC

Figure 6: (a) Recall for an increasing support size, (b) Recall using the MRSC algorithm for the support size varying from 2 to L (L is varied), (c) Query time against increasing support size, (d) Query time for the MRSC algorithm for support size varying from 2 to L. The experiment used 1M SIFT descriptors and the performance averaged over 1K query descriptors.

using 1K SIFT descriptors for two cases (i) using the OST generated hash codes, and (ii) using the MRSC algorithm for the active support size varying from 2 to L; recall that the latter experiment is required in case an empty hash bucket is encountered. Figure 6(a) and Figure 6(b) plot both these cases respectively. Figures 6(c) and 6(d) plot the average query times for both the cases.

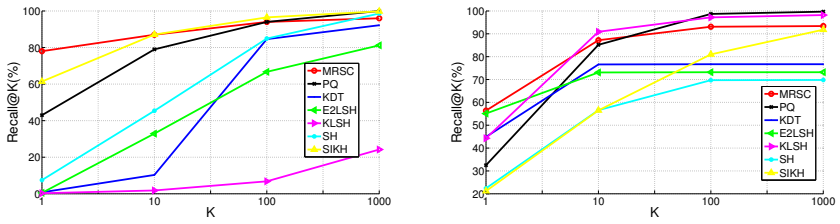
As is clear from the plots, an active set size in the range of 4–7 provides the best recall and the best query time. The exact match of the query SCT with the ground truth happens less than 30% of the time and thus the recall as shown in Figure 6(a) is low. On the other hand, using multiple regularizations help address the issue of empty hash buckets and thus improving the recall as the size of the support increases, as shown in Figure 6(b). Although the query time goes down as the size of the hash code increases as shown in Figure 6(c), it increases after a certain point (L=5) in the MRSC algorithm. As suggested from the above plots, we decided to choose the L value in the range of 2–5 in this paper. A point to note here is that the recall goes low when L=2. This is because, there will be too many hash collisions and the number of coefficients compared against to resolve the collision is too less. Although the above parameter estimation was done on the SIFT dataset, we used the same regularization range for our other datasets and was found to provide good performance.

8.5 Recall Experiments

In this subsection, we compare the recall performance of the MRSC algorithm (cyclic MRSC) against the state-of-the-art techniques. We chose six other popular NN techniques to compare against: (i) Product Quantization (PQ) algorithm [24], (ii) KD-Trees (KDT), (iii) Euclidean Locality Sensitive Hashing (E2LSH) based on [10], (iv) Kernelized LSH (KLSH) [29], (v) Spectral Hashing (SH) [50] and (vi) Shift Invariant Kernel Hashing (SIKH) [42]. Spectral hashing used 40 bit hash codes as it was observed that higher number of bits reduced the recall performance; which is expected as SH internally uses a PCA step, the sensitivity of the higher order bits increases as the length of the hash code increases as more and more basis are included corresponding to smaller eigenvalues. We used 40 bits for SIKH as well. Other algorithms used the default parameters as provided by their implementations. For KLSH, we used the RBF kernel and used 1K descriptors randomly selected from the dataset for learning the kernel. As for the PQ algorithm, we used the IVFADC variant as it seemed to show the best performance both in speed and accuracy.

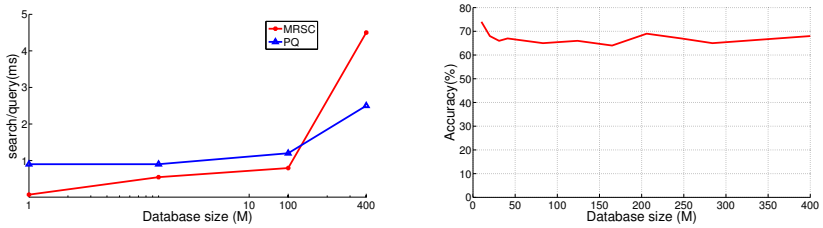
Recall on SIFT: We used approximately 2M SIFT descriptors as the database and 1K query points. The average recall performance after 10 trials for an increasing number of retrieved points is shown in Figure 7(a). As is clear, MRSC algorithm outperforms all the algorithms in recall, especially for recall@1. Although, SIKH performs very close to MRSC, the performance of other algorithms especially the KLSH was found to be very low. We found that the selection of the training set for building the kernel had a major impact on the performance of this algorithm. Thus, we show the best performance of KLSH that we observed from 10 different kernel initializations.

Recall on Spin Images: We used approximately 1M spin images (400D) as the database and 1K descriptors, excluded from the dataset, as the query points. Since the KDT algorithm needs to store all the data points in memory, we did a dimensionality reduction of these descriptors to 40D descriptors through a PCA projection as suggested in [25]. The average recall performance after 10 trials is shown in Figure 7(b) for an increasing number of retrieved points. As we saw with the SIFT dataset, MRSC seems superior to all the other algorithms in the quality of retrieved points.



(a) SIFT (b) Spin Images

Figure 7: Recall@K for SIFT (left) and (right) compared against the state of the art methods. The comparisons were carried out on 1M points from the SIFT and spin image datasets respectively, and the recall computed for a query size of 1K descriptors averaged over 10 trials.



(a) (b)

Figure 8: 8(a) plots the search time per query for increasing database size, 8(b) plots the recall@1 for increasing database size.

8.6 Webscale Datasets

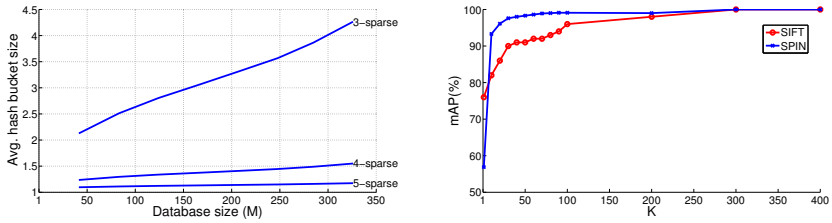
Retrieval Speed: Scalability to large datasets is an important property that any NN algorithm should posses if targeted to work at webscale. In this subsection, we show NN retrieval performance of MRSC (cyclic) against an increasing database size. For this experiment we used the 400M SIFT dataset mentioned previously. The dataset was gradually incremented from 1M towards 400M, each time 1K descriptors were randomly chosen from the dataset (and thus excluding from the database) and recall@1 computed. Figure 8(a) plots the average search time per query for our algorithm compared against the PQ algorithm. Since the optimized implementation of the PQ algorithm is licensed, we do the comparison against the results reported in [24] using the same hardware platform as suggested by the authors. Clearly, as seen from the plot, our implementation is rivals the state-of-the-art in search speed.

Recall Performance: Next, we compared the recall@1 performance of MRSC against an increasing database size. As increasing the database leads to increased hash table collisions, the purpose of this experiment was to analyze how good the approximate sparse Euclidean distance computation was. We used the same 400M SIFT dataset and did the recall@1 for 1K descriptors by gradually increasing database size as was done in the last experiment. Figure 8(b) shows the result. Even though there is a slight drop in the recall as the database size increases, the subsequent drop is minimal and on an average maintains a recall almost close to the recall@1 performance as seen in Figure 7(a). This is explained by the Figure 9(a) which shows that as the database size increases the new points fall onto new hashbuckets such that the relative average increase in the size of the existing buckets is minimal. The 2-sparse plots are omitted from the Figure 9(a) as they grow at a faster rate than support sizes; for example, we have an average bucket size of 27.36 for 40M SIFT descriptors, while this increases to 143.3 for 400M descriptors.

8.7 Precision

Finally, we compare the Precision@K for the SIFT and spin image datasets. For this experiment, we used the same setup as the one for the recall (as discussed in Section 8.5), except that instead of comparing K NNs found by the algorithm against one ground truth, we compared first NN found by the algorithm against K nearest neighbors from the dataset as found using Euclidean linear scans. Figure 9(b) plots the mean average precision (mAP) computed on 1K SIFT and spin image datasets

averaged over 10 trials. As is clear from the plot, the quality of the NNs retrieved by the cyclic MRSC algorithm lies in the top 50 true nearest neighbors for more than 90% of the queries.



(a) (b)

Figure 9: (a)Average hash bucket size against increasing database size, (b) Mean average precision for the SIFT and spin image datasets for 1K query descriptors compared against an increasing number of K-nns found from the dataset using linear scans.

8.8 Robustness to Distortions

SIFT descriptors are commonly employed for finding feature correspondences across images in applications such as motion estimation, 3D reconstruction, etc. The images used to compute the correspondences are generally distorted in multiple ways and thus robust computation of the matching descriptors is essential for the task. In this section, we explicitly evaluate the robustness of the MRSC algorithm to compute SIFT correspondences across images undergoing various commonly seen distortions. For this experiment, we use the SIFT benchmark dataset[5], consisting of eight categories of images; each category containing six images exhibiting a specific type of distortion. The distortion categories are: (i) *BARK*, with images undergoing magnification, (ii) *BIKES*, with fogged images, (iii) *LEUVEN*, with illumination differences, (iv) *TREES*, with Gaussian blur, (v) *BOAT*, with rotation, (vi) *GRAPHIC*, with 3D camera transformation, (vii) *UBC*, granularity introduced by JPEG compression and (viii) *WALL*, with affine transformations. Sample images from each category is shown in Figure 10 in the order of increasing distortions.

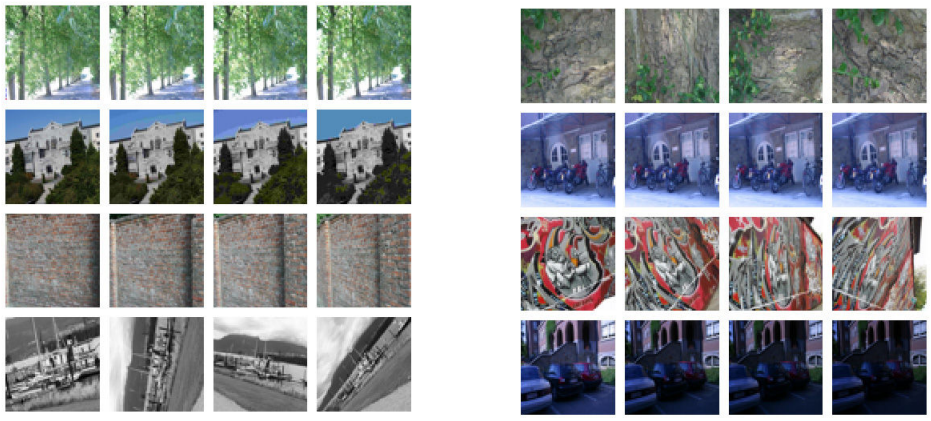


Figure 10: Sample images from each distortion category used to evaluate the recall@1 performance of SIFT descriptors.

SIFT descriptors were computed for each image in a category; the descriptors from the first image in each category was set as the query set, while the database set at step i was set as the descriptors from image i, where i varies from 2 to 6. The recall@1 performance was computed against a ground truth using Euclidean linear scan. As there was too much of distortion in the images for $i = 6$ in each category, we did a slight relaxation of the nearest neighbor criteria such that we declared a candidate point to be a neighbor if it was inside an ϵ-neighborhood of the true neighbor, where we used $\epsilon = 1.12$ (or 90% neighbor). We also pruned out ground truth correspondences that were greater than a threshold in the Euclidean distance,

[5]www.robots.ox.ac.uk/ vgg/research/affine/index.html

as there was a high chance of them being false positives. We compared the performance against three other state-of-the-art algorithms in NN retrieval for the task: KLSH, KDT and PQ, with the same analysis configurations. The result of this experiment is shown in Figure 11(a)–Figure 11(h) with the recall averaged over all the queries in a category. The plots clearly show the robustness of cyclic MRSC against other approaches.

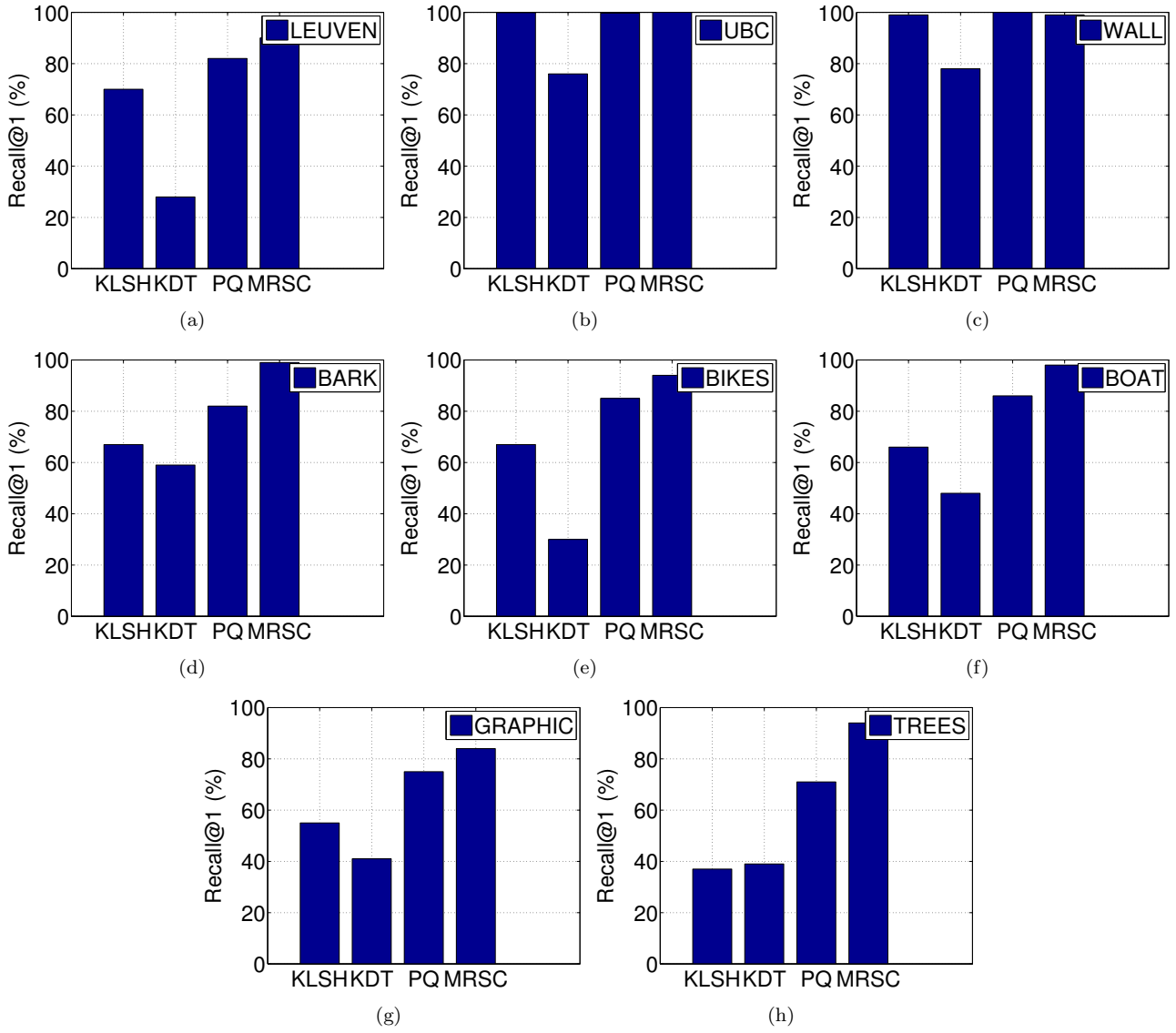


Figure 11: Recall@1 for SIFT descriptors undergoing various image distortions.

9 Conclusion and Future Work

This paper introduced a novel application of sparse coding based on dictionary learning for approximate nearest neighbor searches. A new representation of the data was proposed, which make them directly indexable using an inverted file system. To make this representation deal with noisy data, we proposed an extension of the representation through utilizing the regularization path of the LASSO solution via the LARS algorithm. Extensive experiments were conducted to evaluate our algorithm against web scale datasets and demonstrated superior performance. Going forward, the following ideas are planned to be investigated; rather than performing sparse coding on multiple regularizations, we plan to modify the algorithm such that the dictionary learned is made robust to perturbations in data. This approach leads to a robust dictionary learning

framework. Using a hierarchical sparse coding framework with multiple learned dictionaries of various sizes for further improvising on the robustness is another direction we plan to pursue.

Acknowledgements

We are indebted to three unknown reviewers whose insights and suggestions that has helped us improve this paper. We would like to thank Mr. Duc Fehr, University of Minnesota for helping us with the spin image dataset. This material is based upon work supported in part by the U.S. Army Research Laboratory and the U.S. Army Research Office under contract #911NF-08-1-0463 (Proposal 55111-CI), and the National Science Foundation through grants #IIP-0443945, #CNS-082-1474, #IIP-0934327, #CNS-1039741, and #SMA-1028-076.

References

[1] Aharon, M., Elad, M., Bruckstein, A.: K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. TSP **54**(11), 4311–4322 (2006)

[2] Beyer, K., Goldstein, J., Ramakrishnan, R., Shaft, U.: When is nearest neighbor meaningful? Database Theory pp. 217–235 (1999)

[3] Bohm, C., Berchtold, S., Keim, D.: Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Computing Surveys **33**(3), 322–373 (2001)

[4] Boyer, E., Bronstein, A., Bronstein, M., Bustos, B., Darom, T., Horaud, R., Hotz, I., Keller, Y., Keustermans, J., Kovnatsky, A., Litman, R., Reininghaus, J., Sipiran, I., Smeets, D., Suetens, P., Vandermeulen, D., Zaharescu, A., Zobel, V.: SHREC 2011: robust feature detection and description benchmark. In: Proc. of Eurographics 2011 Workshop on 3D Object Retrieval, pp. 71–78. Eurographics Association (2011)

[5] Broder, A.: On the resemblance and containment of documents. In: Proceedings of Compression and Complexity of Sequences, pp. 21–29. IEEE (1997)

[6] Chen, S., Donoho, D., Saunders, M.: Atomic decomposition by basis pursuit. SIAM Review pp. 129–159 (2001)

[7] Cheng, H., Liu, Z., Hou, L., Yang, J.: Sparsity induced similarity measure and its applications. IEEE Transactions on Circuits and Systems for Video Technology (2012)

[8] Cormen, T.: Introduction to algorithms. The MIT press (2001)

[9] Dalal, N., Triggs, B., Schmid, C.: Human detection using oriented histograms of flow and appearance. In: ECCV. Springer (2006)

[10] Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.: Locality-sensitive hashing scheme based on p-stable distributions. SoCG pp. 253–262 (2004)

[11] Datta, R., Joshi, D., Li, J., Wang, J.: Image retrieval: Ideas, influences, and trends of the new age. ACM Computing Surveys **40**(2), 1–60 (2008)

[12] Donoho, D.: Compressed sensing. Technical Report, Department of Statistics, Stanford University (2004)

[13] Donoho, D.: For most large underdetermined systems of linear equations the minimal L1-norm solution is also the sparsest solution. Communications on Pure and Applied Mathematics **59**(6), 797–829 (2006)

[14] Efron, B., Hastie, T., Johnstone, I., Tibshirani, R.: Least angle regression. Annals of Statistics **32**(2), 407–451 (2004)

[15] Elad, M.: Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing. Springer Verlag (2010)

[16] Elad, M., Aharon, M.: Image denoising via learned dictionaries and sparse representation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 895–900 (2006)

[17] Elad, M., Aharon, M.: Image denoising via learned dictionaries and sparse representation. In: CVPR (2006)

[18] Elad, M., Starck, J., Querre, P., Donoho, D.: Simultaneous cartoon and texture image inpainting using Morphological Component Analysis (MCA). Applied and Computational Harmonic Analysis **19**(3), 340–358 (2005)

[19] Engan, K., Aase, S., Husøy, J.: Multi-frame compression: Theory and design. Signal Processing **80**(10), 2121–2140 (2000)

[20] Gill, P., Wang, A., Molnar, A.: The in-crowd algorithm for fast basis pursuit denoising. TSP **59**(10), 4595–4605 (2011)

[21] Gromov, M.: Monotonicity of the volume of intersection of balls. Geometrical aspects of functional analysis pp. 1–4 (1987)

[22] Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. ToC pp. 604–613 (1998)

[23] Jegou, H., Douze, M., Schmid, C.: Hamming embedding and weak geometric consistency for large scale image search. In: ECCV (2008)

[24] Jegou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. PAMI **33**(1), 117–128 (2011)

[25] Johnson, A.: Spin-images: A representation for 3D surface matching. Ph.D. thesis, Carnegie Mellon University (1997)

[26] Kim, S.J., Koh, K., Lustig, M., Boyd, S., Gorinevsky, D.: An interior-point method for large-scale l1-regularized least squares. Selected Topics in Signal Processing **1**(4), 606–617 (2007)

[27] Klenk, G.: A sparse coding based similarity measure. In: DMIN (2009)

[28] Knuth, D.: The art of computer programming. Vol. 3, Sorting and Searching. Addison-Wesley Reading, MA (1973)

[29] Kulis, B., Grauman, K.: Kernelized locality-sensitive hashing for scalable image search. In: ICCV (2009)

[30] Lee, H., Battle, A., Rajat, R., Ng, A.: Efficient sparse coding algorithms. In: NIPS (2007)

[31] Liu, W., Wang, J., Kumar, S., Chang, S.: Hashing with graphs. In: ICML (2011)

[32] Lowe, D.: Distinctive image features from scale-invariant keypoints. IJCV **60**(2), 91–110 (2004)

[33] Mairal, J., Bach, F., Ponce, J., Sapiro, G.: Online Dictionary Learning for Sparse Coding. In: ICML (2009)

[34] Mairal, J., Bach, F., Ponce, J., Sapiro, G.: Online learning for matrix factorization and sparse coding. JMLR **11**, 19–60 (2010)

[35] Mairal, J., Bach, F., Ponce, J., Sapiro, G., Zisserman, A.: Supervised dictionary learning. Arxiv preprint arXiv:0809.3083 (2008)

[36] Mairal, J., Yu, B.: Complexity analysis of the lasso regularization path. In: arXiv:1205.0079v1 [stat.ML] (2012)

[37] Malioutov, D., Cetin, M., Willsky, A.: Homotopy continuation for sparse signal representation. In: ICASSP. IEEE (2005)

[38] Mallat, S., Zhang, Z.: Matching pursuits with time-frequency dictionaries. TSP **41**(12), 3397–3415 (1993)

[39] Marcellin, M., Gormish, M., Bilgin, A., Boliek, M.: An overview of JPEG-2000. In: Data Compression Conference, pp. 523–541. IEEE (2000)

[40] Mori, G., Belongie, S., Malik, J.: Shape contexts enable efficient retrieval of similar shapes. CVPR (2001)

[41] Murray, J., Kreutz-Delgado, K.: Sparse image coding using learned overcomplete dictionaries. Machine Learning for Signal Processing pp. 579–588 (2004)

[42] Raginsky, M., Lazebnik, S.: Locality-sensitive binary codes from shift-invariant kernels. In: NIPS (2009)

[43] Salakhutdinov, R., Hinton, G.: Semantic hashing. International Journal of Approximate Reasoning **50**(7), 969–978 (2009)

[44] Shakhnarovich, G., Viola, P., Darrell, T.: Fast pose estimation with parameter-sensitive hashing. In: ICCV. IEEE (2003)

[45] Sivic, J., Zisserman, A.: Video Google: A text retrieval approach to object matching in videos. In: Proceedings of the IEEE International Conference on Computer Vision, vol. 2, pp. 1470–1477 (2003)

[46] Snavely, N., Seitz, S., Szeliski, R.: Photo tourism: exploring photo collections in 3D. In: TOG, vol. 25, pp. 835–846. ACM (2006)

[47] Tibshirani, R.: Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological) pp. 267–288 (1996)

[48] Torralba, A., Fergus, R., Weiss, Y.: Small codes and large image databases for recognition. CVPR (2008)

[49] Wang, J., Yang, J., Yu, K., Lv, F., Huang, T., Gong, Y.: Locality-constrained linear coding for image classification. In: CVPR. IEEE (2010)

[50] Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: NIPS (2009)

[51] Yang, J., Yu, K., Gong, Y., Huang, T.: Linear spatial pyramid matching using sparse coding for image classification. In: CVPR. IEEE (2009)

[52] Yang, J., Zhang, Y.: Alternating direction algorithms for l1-problems in compressive sensing. SIAM Journal on Scientific Computing **33**, 250–278 (2011)

[53] Zepeda, J., Kijak, E., Guillemot, C.: SIFT-based local image description using sparse representations. In: MMSP, pp. 1–6. IEEE (2009)