From High Definition Image to Low Space Optimization

Micha Feigin, Dan Feldman, and Nir Sochen

Abstract. Signal and image processing have seen in the last few years an explosion of interest in a new form of signal/image characterization via the concept of sparsity with respect to a dictionary. An active field of research is dictionary learning: Given a large amount of example signals/images one would like to learn a dictionary with much fewer atoms than examples on one hand, and much more atoms than pixels on the other hand. The dictionary is constructed such that the examples are sparse on that dictionary i.e each image is a linear combination of small number of atoms.
This paper suggests a new computational approach to the problem of dictionary learning. We show that smart non-uniform sampling, via the recently introduced method of *coresets*, achieves excellent results, with controlled deviation from the optimal dictionary. We represent dictionary learning for sparse representation of images as a geometric problem, and illustrate the coreset technique by using it together with the K−SVD method. Our simulations demonstrate gain factor of up to 60 in computational time with the same, and even better, performance. We also demonstrate our ability to perform computations on larger patches and high-definition images, where the traditional approach breaks down.

Keywords: Sparsity, dictionary learning, K−SVD, coresets

1 Introduction

One of the major problems in image processing is image characterization. By image characterization we mean a system that gets a two-dimensional function or, in the discrete case, a matrix with non-negative entries, as a query and provides an answer whether or not this function/matrix is an image. Other option is that the system provides a probability measure on the space of all two-dimensional such functions/matrices. We are still far from achieving this ultimate goal, yet few breakthroughs where recorded since the inception of image processing as a branch of scientific research. Many characterization, in the past, used the decay rate of the coefficients of certain transformations. That led to a characterization in a linear space of functions. In the last decade a new approach that involves redundant representations and sparsity seems promising. In this framework, a signal is represented again as a superposition of signals. But unlike the representation with a basis of a linear space, the number of basic signals (a.k.a. atoms) in this new approach exceeds the dimension of the signal such that a given signal may have many different representations. Uniqueness is achieved

only for a subset of signals which can be represented with a limited number of atoms, called sparse signals. For this class of signals the sparsest representation is unique. This approach shifts the focus of attention from the general law of decay of coefficients to the outliers of such behavior, namely the large coefficients of such an expansion. The class of sparse signals does not form a linear space which reflects the non-linearity of the set of images. At the same time, we still use linear techniques which helps a lot in practice.

The sparsity approach has appealing features for image processing, but it suffers from few problems. First it is clear that sparsity is a notion which is attached to a given dictionary. Clearly, there is no one universal dictionary that can represent any image in a sparse way. This calls upon the need to construct dictionary for each class of images or for each application. Constructing a dictionary for a large number of images from the same class/application goes under the name dictionary learning and is an active field of research. This is the main topic of this paper, and we demonstrate our ideas on the K−SVD method [3]. Second is the very extensive use of computational time and in memory space. Because of the prohibitive computational time and the numerical instabilities in computing with large size matrices, sparsity techniques are applied to small images only. In fact, 8×8 to 16×16 is the most common sizes in image processing. It means that these are patches of images rather than images themselves. Moreover, one may wish to construct dictionaries for the same class of images. Using, implicitly, the approximate self-similarity nature of images it is costumed to use the patches of an image as a class of similar patches and to construct a dictionary per image. Here, again, the curse of limited space and time interfere and high definition images (1024×1024 say) have a huge number of patches of such a small size which makes the dictionary learning task computationally prohibitive. This paper brings the spell of coresets to cure the curse of space and time limitations. Informally, a coreset C for Y is a compressed representation of Y that well approximates the original data in some problem-dependent sense. The coreset C is used to give approximate answers for queries about Y.

We show that the optimization problem can be solved on the coreset, which is much smaller, without sacrificing too much accuracy. Corsets techniques were first introduced in the computational geometry field, and in the recent years used to solve some well known open problems in computer science and machine learning. The subject became more mature theoretically in the last few years. Corsets present a new approach to optimization in general and have huge success especially in tasks which use prohibitively large computation time and/or memory space. In particular, coresets suggest ways to use existing serial algorithms for distributed (parallel) computing, and provide solutions under the the streaming model, where the space (memory) for solving the problem at hand is significantly smaller than its input size.

2 Coresets for Dictionaries

Approximation algorithms in computational geometry often make use of random sampling, feature extraction , and ϵ-samples. Coresets can be viewed as a general concept that includes all of the above, and more. See a comprehensive (but not so updated) survey on this topic in [1]. Coresets have been the subject of many recent papers (see references in [1]) and several surveys [2, 4].

In our context, the input is an $d \times n$ matrix Y that represents n points in $\mathbb{R}^d$, and we consider a $d \times k$ matrix D, whose columns represent k points in $\mathbb{R}^d$. The matrix D is called a *dictionary*. Typically n is much larger than k, and k is much larger than d. Let $\text{cost}(\cdot, \cdot)$ be some function of Y and D. We interpret $\text{cost}(Y, D)$ as the result of a query D on a matrix Y.

2.1 k-dictionary queries

Let D be a dictionary. The column vectors of the matrix D are called the *points of* D. We write $z \in D$ if z is one of the columns of D. Let y be a point (vector) in $\mathbb{R}^d$, and let $\text{err}(y, D)$ be a non-negative real function that represents the error of approximating y by D.

Through the rest of this section we won't assume anything further about the function $\text{err}(\cdot, \cdot)$. Following some possible definitions of err that are relevant for this paper, when D is a $d \times k$ matrix:

1. $\text{err}(y, D) = \min_{x \in \mathbb{R}^k} \|Dx - y\|_2$ is the Euclidean distance between the point y and the subspace that is spanned by the columns of D.
2. $\text{err}(y, D) = \min_{x \in \{e_i\}} \|Dx - y\|_2$ is the distance between y and its closest point of D. Here, $\{e_i\} = \{e_i\}_1^k$ denotes the standard base of $\mathbb{R}^k$.
3. $\text{err}(y, D) = \min_{x \in \mathbb{R}^k, \|x\|_0 = 1} \|Dx - y\|_2$ is the distance between y and the closest line that intersects both the origin of $\mathbb{R}^d$ and a point in D. Here $\|x\|_0$ denotes the number of non-zeros entries of x.
4. For an integer $j \geq 0$, $\text{err}_j(y, D) = \min_{x \in \mathbb{R}^k, \|x\|_0 \leq j} \|Dx - y\|_2$ is the distance between y and its closest subspace over the set of $O\left(\binom{k}{j}\right)$ subspaces that are spanned by at most j points of D.

More generally, we can replace $\|Dx - y\|_2$ by $\|Dx - y\|_p^q$ for $p, q \geq 0$ in the above examples.

Problem 1 (k-Dictionary Query). The input to this problem is an integer $k \geq 1$ and a $d \times n$ matrix Y. For a given (query) $d \times k$ dictionary D, the desired output is $\text{cost}(Y, D) = \sum_{y \in Y} \text{err}(y, D)$.

Suppose that, given $y \in Y$ and a $d \times k$ dictionary D, the error $\text{err}(y, D)$ can be computed in time T. Then $\text{cost}(Y, D)$ can be computed in $O(Tn)$ time and using $O(dn)$ space. Here, "space" means memory, or number of non-zeros entries. In this paper we wish to pre-process the input Y in such a way that an ε-approximation of the output $\text{cost}(Y, D)$ could be computed in time $O(Tc)$ and space $O(dc)$, where c is sub-linear (actually, independent) in n. To this end, we introduce the concept of ε-coreset for the k-dictionary problem.

2.2 Coreset for a single k-dictionary query

A matrix C is called a *weighted matrix* if every column $y \in C$ is associated with a weight $w(y) \geq 0$. For a weighted matrix C and a dictionary D, we define

$$\text{cost}(C, D) = \sum_{y \in C} w(y)\text{err}(y, c).$$

A (non-weighted) matrix Y is considered a weighted matrix with $w(y) = 1$ for every $y \in Y$.

Definition 1 (ε-coreset). *Let Y be an $d \times n$ matrix, and $\mathcal{D}$ be a set of $d \times k$ dictionaries. Let C be a weighted $d \times c$ matrix. We say that C is an* ε-coreset *for Y if, for every $D \in \mathcal{D}$, we have*

$$(1 - \varepsilon)\text{cost}(Y, D) \leq \text{cost}(C, D)| \leq (1 + \varepsilon)\text{cost}(Y, D). \tag{1}$$

For example, it is easy to verify that Y is an ε-coreset of itself. However, an ε-coreset C is efficient if $c << n$. In this case, $\text{cost}(C, D)$ can be computed in $Tc << Tn$ time using only $cd << nd$ space.

Algorithm `Coreset`(Y, D_0, c).
Input: a $d \times n$ matrix Y, an integer $c \geq 1$, and a matrix D_0 (of arbitrary size).
Output: a weighted $c \times d$ matrix C that satisfies Theorem 1.

Pick a non-uniform random sample $S = \{s_1, s_2, \cdots, s_c\}$ of c i.i.d. columns from Y, where $y \in Y$ is chosen with probability proportional to $\text{err}(y, D_0)$. That is, for every $s \in S$ and $y \in Y$, the probability that $s = y$ is

$$\text{pr}(y) = \frac{\text{err}(y, D_0)}{\sum_{y \in Y} \text{err}(y, D_0)} = \frac{\text{err}(y, D_0)}{\text{cost}(Y, D_0)}.$$

Return the weighted matrix C whose columns are the vectors of S (in some arbitrary order), where each $y \in C$ is weighted by

$$w(y) = \frac{1}{c \cdot \text{pr}(y)}. \tag{2}$$

The following lemma can be easily proved using Chernoff-Hoeffding's inequality.

Lemma 1. *Let Y be a $d \times n$ matrix, and $\delta, \varepsilon > 0$ be. Let C be the output of the algorithm* `Coreset` *with input parameters Y, D_0 and*

$$c \geq \frac{10 \ln(1/\delta)}{\varepsilon^2}.$$

Let D be a fixed $d \times k$ dictionary. Then, with probability at least $1 - \delta$,

$$|\text{cost}(Y, D) - \text{cost}(C, D)| \leq \varepsilon \text{cost}(Y, D_0) \cdot \max_{y \in Y} \frac{\text{err}(y, D)}{\text{err}(y, D_0)}.$$

Corollary 1. *Let $b \geq 1$ be an integer and D_0 be a dictionary, such that for every $D \in \mathcal{D}$:*

(i) $\text{cost}(Y, D_0) \leq b \cdot \text{cost}(Y, D)$.
(ii) $\text{err}(y, D) \leq \text{err}(y, D_0)$ *for every* $y \in Y$.

Put $\varepsilon, \delta > 0$. Let C be the weighted matrix that is returned by the algorithm `Coreset` *with input parameters $c \geq 10b^2 \ln(1/\delta)/\varepsilon^2$ and D_0. Then, for a fixed dictionary $D \in \mathcal{D}$ (which is independent of C), we have*

$$(1-\varepsilon)\text{cost}(Y, D) \leq \text{cost}(C, D) \leq (1+\varepsilon)\text{cost}(Y, D),$$

with probability at least $1 - \delta$.

Proof. We have $\text{cost}(Y, D_0) \leq b\text{cost}(Y, D)$ by property (i). Replacing ε with ε/b in Lemma 1 yields

$$\begin{aligned} |\text{cost}(Y, D) - \text{cost}(C, D)| &\leq (\varepsilon/b)\text{cost}(Y, D_0) \max_{y \in Y} \frac{\text{err}(y, D)}{\text{err}(y, D_0)} \\ &\leq \varepsilon \text{cost}(Y, D) \max_{y \in Y} \frac{\text{err}(y, D)}{\text{err}(y, D_0)} \leq \varepsilon \text{cost}(Y, D), \end{aligned}$$

where the last inequality follows from property (ii).

2.3 Coreset for all k-dictionary queries

In order to have an ε-coreset for a set $\mathcal{D}$ of more than one dictionary, there are still two problems that remain to be solved. Firstly, we need to compute D_0 that satisfies Properties (i) and (ii) of Corollary 1 with sufficiently small b. This will be handle in the next section for our specific applications. Secondly, Definition 1 of ε-coreset demands that C will approximate $\text{cost}(C, D)$ *simultaneously* for *every* $D \in \mathcal{D}$. However, Corollary 1 holds, with probability at least $1 - \delta$, only for a fixed dictionary $D \in \mathcal{D}$, i.e, a single query. If the size of $\mathcal{D}$ is finite, we can replace δ with $\delta/|\mathcal{D}|$ in Corllary 1 and use the union bound to obtain an ε-coreset for Y of size

$$c = O\left(\frac{\ln(|D|)b^2 \ln(1/\delta)}{\varepsilon^2}\right). \tag{3}$$

However, in the applications of this paper the size of $\mathcal{D}$ is infinite. In this case, we use the result of [7] that is based on PAC-learning theory. Roughly speaking, the result states that to obtain an ε-coreset, it suffices to replace the term $\ln(|D|)$ in (3) by some dimension v that represents the complexity of the set $\mathcal{D}$. This dimension is similar to the classic notion of VC-dimension that is used in machine learning [14]. Usually v is proportional to the number of parameters that are needed to represent a dictionary D of $\mathcal{D}$, which is, in the general case, the number dk of entries in the matrix D.

Theorem 1. *Let Y be a $d \times n$ matrix, $\varepsilon, \delta, b > 0$ and $v \leq O(dk)$ be the dimension that corresponds to a set $\mathcal{D}$ of $d \times k$ matrices. Let D_0 be a matrix as defined in Corollary 1. Let C be the weighted matrix that is returned by the algorithm* `Coreset` *with input parameters $c \geq v10b^2 \ln(1/\delta)/\varepsilon^2$ and D_0. Then, with probability at least $1 - \delta$, C is an ε-coreset of Y.*

3 Example Application: Approximating the Optimal Dictionary

3.1 The k-dictionary problem.

In this section we consider the following problem. The input is a parameter $k \geq 1$, a $d \times n$ matrix Y, and an error function $\text{err}(\cdot, \cdot)$. The output is a $d \times k$ dictionary D^* that minimizes $\text{cost}(Y, D) = \sum_{y \in Y} \text{err}(y, D)$ over a given set $D \in \mathcal{D}$. For $\varepsilon > 0$, a $(1+\varepsilon)$-approximation for this problem is a matrix D such that $\text{cost}(Y, D) \leq (1+\varepsilon)\text{cost}(Y, D^*)$.

For the error function 1, 2 and 3 in the beginning of Section 2.1 above, provable $(1+\varepsilon)$-approximation algorithms are provided in [7–9], using coresets techniques.

For the more general and popular case in the context of sparse dictionaries where $j > 2$ (the fourth error function in Section 2.1) D^* minimizes the distances to a set of $\binom{k}{j}$ subspaces under constraints. Very little is known about minimizing distances of points to multiple affine subspaces (that are neither points or lines). For example, the problem of computing a pair of planes in three dimensional space that minimize the sum (or sum of squared) distances from every point to its closest plane is an open problem. This holds even for a corresponding constant factor approximation.

In fact, it is not clear how to compute the closest distance to a set of subspaces efficiently even for a *given* dictionary D and a point $y \in y$ in time that is polynomial in j. This is equivalent to answer Problem 1 for a matrix Y with a single column y. Nevertheless, a lot of heuristics have been suggested over the years for approximating distance to subspaces (called pursuit algorithms, see a detailed description of these methods in [3]), for approximating points by subspaces in general (see [6] and references therein), and for the k-dictionary problem in particular (see references in [3]).

3.2 Coreset for the k-dictionary problem

We prove that, under natural assumptions on the input matrix Y and the set of candidate dictionaries $\mathcal{D}$, we can choose input parameters c and D_0 such that the algorithm `Coreset`(Y, D_0, c) from Section 2.2 returns a small ε-coreset C of Y.

There are two issues that need to be resolved: Firstly, it is not clear how to compute $b \geq 1$ and the input D_0 from Theorem 1 that will satisfy the two properties of Corollary 1. Note that the size c of the output coreset depends on b, so we would like to choose D_0 such that b is small. Secondly, we need to compute the dimension v of the set $\mathcal{D}$ of all possible $d \times k$ dictionaries. Recall that by Theorem 1, v is required in order to compute c.

In order to satisfy property (i), we use the fact that in our experiments on both the synthetic and the real data, the variance of entries in every column vector y of the input matrix Y is generally small. This is because y usually represents a small block of pixels in an image. Letting D_0 be the $d \times 1$ matrix

(vector) of ones guarantees that $\mathrm{err}(y, D_0)$ is not larger than the variance of y. Hence $\mathrm{cost}(Y, D_0) = \sum_{y \in Y} \mathrm{err}(y, D_0)$ is the sum of these variances which is usually not much larger than $\mathrm{cost}(Y, D)$ for every $D \in \mathcal{D}$. This satisfies property (i). Adding this vector D_0 for every possible output dictionary $D \in \mathcal{D}$ satisfies property (ii) since in this case $\mathrm{err}(y, D) \leq \mathrm{err}(y, D_0)$ for every $y \in Y$.

Another option that will satisfy the above two properties is relevant when running a heuristic that uses several iterations in order to compute D^*. In this case, we may run the heuristic on the original input data Y only for a single iteration, and choose D_0 to be the returned dictionary. Property (i) will hold under the assumption that the ratio b between the initial dictionary and the rest of the dictionaries that will be computed by the heuristic is not very large. Property (ii) will be satisfied under the assumption that $\mathrm{err}(y, \cdot)$ is highest on the first iteration of the heuristic, for every column y of Y.

We chose the first option (where $D_0 = (1, \cdots, 1)$), for practical reasons (due the simplicity of implementation), and for theoretical reasons (since the second option assumes that the heuristic is based on iterations). Interestingly enough, we found out that existing heuristics (such as the K−SVD algorithm in [3]) already add such a constant vector D_0 for every dictionary that they output, for different reasons. We summarize our decision and its justification in the following theorem.

Theorem 2. *Let $D_0 = (1, \ldots, 1)^T$ be the d-dimensional vector of ones. Let $\mathcal{D}$ be a set of $d \times k$ dictionaries, such that each $D \in \mathcal{D}$ contains D_0, and suppose that $OPT = \min_{D \in \mathcal{D}} \mathrm{cost}(Y, D) > 0$. Let v denote the dimension of $\mathcal{D}$.*

Let Y be a $d \times n$ matrix, and define $b = \mathrm{cost}(Y, D_0)/OPT$. Let $c \geq 10vb^2 \ln(1/\delta)/\varepsilon^2$ for some $\varepsilon, \delta > 0$. Then `Coreset`(Y, D_0, c) returns, with probability at least $1 - \delta$, an ε-coreset of Y.

Determining c. Although the set of dictionaries that a heuristic tests (queries) during its running time is finite, we still cannot use the union bound with Corollary 1, since these dictionaries depend on the input coreset C. However, it is reasonable to assume that, for a given Y, not all the possible $d \times k$ matrices D have a positive probability (over the randomness of C) to be queried by the algorithm. That is, we believe that the dimension v of the candidate set $\mathcal{D}$ is significantly smaller than dk, but a more involved theoretical analysis of the corresponding heuristic is needed. Also, it is not clear how to compute b in Theorem 2. Practically, we will simply apply the algorithm `Coreset` with a value of c that is determined by the available memory and time at hand. Hence, the parameters δ, ε, b and v that are defined in Theorem 2 will be used only for the theoretical analysis. Nevertheless, they guarantee that the required sample size c is generally "small" for a reasonable values of δ and ε. The theoretical bound on c also teaches us about the relation between the input parameters. For example, the fact that c in Theorem 1 is independent of n implies that the ratio between the computation time of $\mathrm{cost}(Y, D)$ and $\mathrm{cost}(C, D)$ converges to infinity for asymptotically large n, while the error that is introduced by the coresets

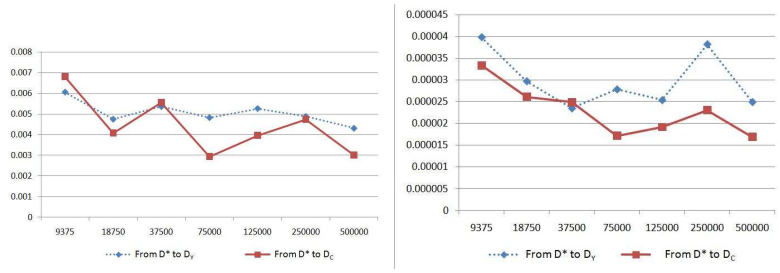
remains approximately the same. Indeed, we observe these two phenomenas in our experiments; see Fig. 1(a) and 2(a).

4 Experimental Results

Hardware We ran the experiments on a standard personal modern Laptop. Namely, IBM Lenovo W500, as provided by the manufacturer, without additional hardware. In particular, we used the CPU "Intel Core 2 Duo processor T9600 (2.80 GHz)" with 2GB memory. See manufacturer's website (http://www-307.ibm.com/pc/support/site.wss/document.do?lndocid=MIGR-71785) for exact hardware details.
Software The operation system that we used is "Windows Vista Business" and the Matlab version is 2010b. For the K−SVD and OMP algorithms, we used the implementation of Rubinstien that was generously uploaded on the Internet [13]. This implementation was used as a "black box" without changing a line of code. The time and space improvements are therefore only due to the replacing of the original input matrix Y with its coreset.

4.1 Synthetic data

As in previously reported works [10, 11, 3], we first try to construct coresets of synthetic data. In [3] it was shown how the K−SVD algorithm approximated the original dictionary D^* that generated a synthetic data matrix Y. In the following experiments we replace Y by its (usually much smaller) coreset C, and compare the results of applying K−SVD on C instead of Y. The construction of C is done using algorithm `Coreset` with D_0 and different values of c, as defined in Theorem 2. The construction of the generative dictionary D^* and the input matrix Y was based on the suggested experiments in [3]. *Generating the dictionary D^* and the matrix Y.* A random (dictionary) matrix D^* of size $d \times k = 20 \times 50$ was generated with i.i.d. uniformly distributed entries. Each column was normalized to a unit norm. Then, a $20 \times n$ matrix Y was produced for different values of n. Each column y of Y was created using a linear combination D^*x of $\|x\|_0 = j = 3$ random and independent different columns of D^*, with uniformly distributed i.i.d. coefficients. White Gaussian noise with varying signal-to-noise ratio (SNR) $\sigma = 20$ was added to the resulting vector D^*x. That is, $Y = D^*X + N$ where N is a matrix that represents the Gaussian noise in each entry, and every column x of X corresponds to a column vector y in Y as defined above. We run the experiment with 11 different assignments for n, that were approximately doubled in every experiment: from $n = 585$ to $n = 500,000$. For every such value of n, 50 trials were conducted, when in every trial new dictionary D^* and matrices Y and X were constructed.
Applying K−SVD *on Y.* We run the K−SVD implementation of [13], where the maximum number of iterations was set to 40. The rest of parameters were the defaults of the implementation in [13]. We denote the output dictionary by D_Y.
Generating the coreset C. We implemented and run the algorithm `Coreset`(c, D_0)



(a) (b)

Fig. 1. Comparison of the differences between the dictionaries D_Y D_C and D^* over the number n of rows in the matrix Y. The dictionaries D_Y, D_C are respectively the dictionaries that were constructed using the original matrix Y, and its coreset C. The original generator dictionary of Y is denoted by D^*.

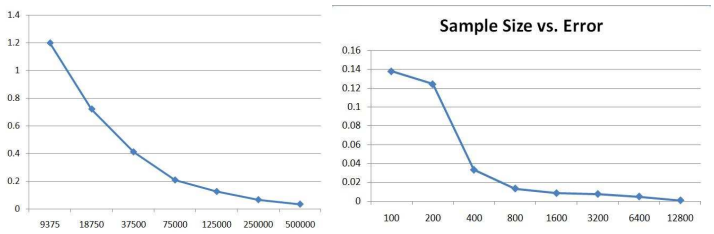
from Section 2.2 on the input matrix Y where the size of the coreset was set to $c = 5000$. The parameter D_0 was always set to be the column vector of d ones. See Section 3.2 for more details.

Applying K−SVD *on* C. We called to the K−SVD algorithm using the same parameters as the above call for Y, except for the maximum number of iterations. After setting the number of iterations to 40 for the input C (as in the runs on Y), we got results that are only slightly worse than on Y, but significantly faster (up to 100 times). We therefore decided to sacrifice time in order to get better results, and used 120 iterations on the K−SVD with the input C. We denote the output dictionary by D_C.

Approximating the sparse coefficients matrix. In order to approximate the entries of the matrix X, we used the OMP heuristic as defined in [12] and implemented in [13]. The objective of OMP is to minimize $\|Y - D_Y X_Y\|_F$ for the given dictionary D_Y and the input matrix Y, over every matrix X_Y whose columns are sparse ($\|x\|_0 = j = 3$ for every column $x \in X_Y$). This is done by minimizing $\|y - D_Y x\|_F$ for every column $y \in Y$ (one by one) over the set of j-sparse vectors x. Similarly, we computed X_C that suppose to minimize $\|Y - D_C X_C\|$ using the OMP heuristic, as done for Y and D_Y.

Measurement. To measure how close D_Y is to D^*, compared to the difference between D_C and D^*, we used the same error measurement Distance(D, D^*) that was used in the original K−SVD paper [3], and implemented in [13]. The computation of Distance(D, D^*) for two dictionaries D and D^* is done by sweeping through the columns of D^* and finding the closest column (in distance) in the computed dictionary D, measuring the distance via $1 - |d_i^T \tilde{d}_i|$, where d_i is a column in D^* and $\tilde{d}_i$ is its corresponding element in the recovered dictionary D. The average distance is denoted by Distance(D, D^*). That is, Distance(D, D^*)

is the sum of distances over every i, $1 \leq i \leq k$, divided by k. *The Results* In



(a) (b)

Fig. 2. (a) Ratio between running times of K−SVD over coreset C and original input Y. **(b)** The distance between the approximated dictionary D_C and the generating dictionary D for different sizes of coreset C.

Fig. 2(a) we compare the difference (the y-axis) between the dictionaries (the two lines) for different values of n (the x-axis). For example, the dotted line show the average value, for every assignment of n, of Distance(D_C, D^*) over the 50 trials , between the generation dictionary D^* and the dictionary that returned when running K−SVD with the input matrix Y. The variance over the sets of 50 experiments that corresponds to the average in Fig 2(a) is shown in Fig. 2(b). The comparison between the running times appears in Fig 2. The x-axis shows the values of n as in Fig. 2, while the y-axis is the ratio between the running time of constructing D_Y, the dictionary of Y, and the running time of constructing D_C, the dictionary of C. The construction time for D_C is the sum of the time it took to construct the coreset C from Y, and the time for constructing D_C from C.

Discussion In Fig. 2(a) we see that the coreset is usually good at least as the original set for reconstructing the generating dictionary D^*. By Theorem 2, the quality of the coreset C depends on its size c, but not of n. Indeed, the error in Fig. 2 seems to be independent of the value of n. In Fig. 2(b) we see that the results are also more stable on the coreset runs.

Since the size of the coreset is the same ($c = 5000$), the value of n is getting larger, and the running time of the K−SVD algorithm is linear in the rows of the input matrix (c or n), it is not surprising that the ratio between running times grows linearly with the value of n; see Fig. 2(a). For $n = 500K$ in Fig 2(a), the ratio between the running time is approximately 1:30 (0.032). For $n = 1M$ this ratio is approximately 1:60. However due to time and memory constraints we didn't repeat the experiment for $n = 1M$ 50 times.

The role of the sample size c. By Theorem 2, the size c of the coreset C is polynomial in $1/\varepsilon$, where ε represents the desired quality of the coreset. In Fig. 2(b)

we show results for additional set of experiments for a constant $n = 500K$ and different values of the coreset size c. The number of iterations is still 120, and the rest of the parameters remain the same as in the previous experiments. The y-axis is the log of the distance between the dictionaries (base 10) over 50 trials. Indeed, it seems that the error is reduced roughly linearly with the size of c.

(a) (b)

Fig. 3. (a) Noisy Image with SNR= 50. The resulting $PSNR$ is $14.15dB$. **(b)** Denoised image using [5] on the small coreset. The resulting $PSNR$ is ~ 30.9.

4.2 Coresets for High-Definition Images

In [5] it is explained how to apply image denoising using the algorihtm K−SVD. Fortunately, source code was also provided by Rubinstein [13]. We downloaded high-definition images from the movie "Inception"' that was recently released by Warner Bros; see web page "http://collider.com/new-inception-images-posters-christopher-nolan/34058/". We used only one of the images, whose size is $4752 \times 3168 = 15,054,336$ pixels; see Fig. 3. We added a Gaussian noise of $SNR = 50$ which yields a noisy image of $PSNR = 14.15$. Then, we partition the noisy image into 8×8 blocks as explained in [5], and convert the blocks into a matrix Y of approximately $n = 12M$ vectors of dimension $d = 8 \times 8 = 64$. We then hoped to apply the K−SVD as explained in [5] using the default parameters in [13]. However, we got "out of memory" error from Matlab already in the construction of Y.

So, instead, we constructed a coreset C of Y in the streaming model using one pass over Y. In this model, coresets are constructed (using our algorithm `Coreset`) from subsets of columns of Y that are loaded one by one and deleted from memory. When there are too many coresets in memory, a coreset for the union of coresets is constructed and the original coresets are deleted. See details in [8]. After constructing such a coreset C of size $c = 10000$ for *all* the

columns of Y, we apply the K−SVD on the coreset using sparsity $j = 10$, and $k = 256$ atoms, and 40 iterations. The $PSNR$ was increased, on average of 10 experiments, from 14.15 to 30.9, with variance of ~ 0.002, while the average time for constructing the dictionary was 69 seconds with variance of ~ 7.2

5 Conclusions and Further work

We tried to repeat our experiments on real data where Y is partitioned into larger blocks of size $d = 50 \times 50 = 2500$ and an overcomplete dictionary of $k > 3000$. Although the construction of C was fast, the running time of the OMP algorithm (that is used by K−SVD and the denoising procedure for applying the dictionary) is extremely slow when d and k are so large. Besides running time problems, it is noted in [5, 13, 3] that K−SVD does not scale for high dimensional spaces (i.e, large blocks size). We believe that this problem can be solved using recent and more involved coresets techniques of clustering data in high dimensional space, and coresets for the OMP algorithm. We leave this for future papers.

References

1. P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *Journal of the ACM*, 51(4):606–635, 2004.
2. P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Geometric approximations via coresets. *Combinatorial and Computational Geometry - MSRI Publications*, 52:1–30, 2005.
3. M. Aharon, M. Elad, and A. Bruckstein. K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *Signal Processing, IEEE Transactions on*, 54(11):4311–4322, 2006.
4. A. Czumaj and C. Sohler. Sublinear-time approximation algorithms for clustering via random sampling. *Random Struct. Algorithms (RSA)*, 30(1-2):226–256, 2007.
5. M. Elad and M. Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Trans. Image Processing*, 15(12):3736–3745, December 2006.
6. D. Feldman, A. Fiat, D. Segev, and M. Sharir. Bi-criteria linear-time approximations for generalized k-mean/median/center. In *Proc. 23rd ACM Symp. on Computational Geometry (SOCG)*, pages 19–26, 2007.
7. D. Feldman and M. Langberg. A unified framework for approximating and clustering data. In *Proc. 41th Annu. ACM Symp. on Theory of Computing (STOC), to appear*, 2011.
8. D. Feldman, M. Monemizadeh, and C. Sohler. A PTAS for k-means clustering based on weak coresets. In *Proc. 23rd ACM Symp. on Computational Geometry (SoCG)*, pages 11–18, 2007.
9. S. Har-Peled. Low rank matrix approximation in linear time. *Manuscript*, 2006.
10. K. Kreutz-Delgado, J.F. Murray, B.D. Rao, K. Engan, T.W. Lee, and T.J. Sejnowski. Dictionary learning algorithms for sparse representation. *Neural computation*, 15(2):349–396, 2003.

11. S. Lesage, R. Gribonval, F. Bimbot, and L. Benaroya. Learning unions of orthonormal bases with thresholded singular value decomposition. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP'05). IEEE International Conference on*, volume 5. IEEE, 2005.
12. YC Pati, R. Rezaiifar, and PS Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on*, pages 40–44. IEEE, 2002.
13. Ron Rubinstein. Ksvd-box v13. Technical report.
14. V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.