Compressed Singular Value Decomposition for Image and Video Processing

N. Benjamin Erichson, Steven L. Brunton and J. Nathan Kutz
Department of Applied Mathematics
University of Washington
Seattle, WA 98195, USA
erichson@uw.edu, sbrunton@uw.edu, kutz@uw.edu

Abstract

We demonstrate a heuristic algorithm to compute the approximate low-rank singular value decomposition. The algorithm is inspired by ideas from compressed sensing and, in particular, is suitable for image and video processing applications. Specifically, our compressed singular value decomposition (cSVD) algorithm employs aggressive random test matrices to efficiently sketch the row space of the input matrix. The resulting compressed representation of the data enables the computation of an accurate approximation of the dominant high-dimensional left and right singular vectors. We benchmark cSVD against the current state-of-the-art randomized SVD and show a performance boost while attaining near similar relative errors. The cSVD is simple to implement as well as embarrassingly parallel, i.e, ideally suited for GPU computations and mobile platforms.

1. Introduction

The singular value decomposition (SVD) is among the most ubiquitous and powerful methods for data processing in the computational era. However, the emergence of massive data has severely challenged our ability to compute this fundamental matrix decomposition, placing significant constraints on both memory and processing power. In particular, in the context of mobile computing platforms like drones or autonomous underwater vehicles, which provide only very limited compute power, data acquisition is often far outstripping computational resources. In computer vision, for instance, there is a growing demand for higher-resolution imaging, exemplified by 4K video streams. LIDAR (light detection and ranging) also faces a similar growth trajectory as it migrates to the consumer electronics market, since it is expected to be become a standard feature on autonomous vehicles for detecting surrounding objects. Hence, innovations that reduce the computational and storage demands of the SVD are increasingly important, i.e., algorithms that scale with the underlying signal complexity rather than the size of the ambient measurement space.

Randomized algorithms have been demonstrated to be highly viable methods to substantially reduce the computational demands of the SVD [18, 24, 31]. These methods are, in particular, suitable for high-dimensional signals, which feature a low-rank structure. This means, that the intrinsic rank of the data is relatively small compared to the dimension of the ambient measurement space. The idea is to derive a smaller matrix from the high-dimensional input matrix which captures the essential information. This can be done efficiently by using random sampling as a computational strategy. Then, the smaller matrix is used to learn, or extract, the dominant structure from the data.

We demonstrate a simple and computationally efficient two-pass algorithm to compute the approximate low-rank SVD. Specifically, we leverage concepts from compressed sensing to sketch the row space of the input matrix, as illustrated in Figure 1. This approach faithfully preserves

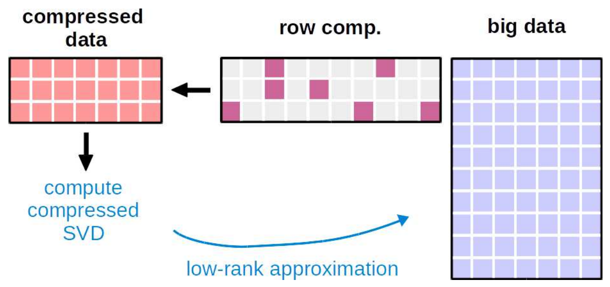


Figure 1: Schematic illustration of the compressed SVD architecture.

the dominant spectral information, and enables an efficient computation which approximates the high-dimensional SVD. The results show that the compressed SVD is, in particular, suited for image and video processing applications. Our work is in closely related to the ideas of Drinea et al. [10] and it has connections to existing randomized and sketched matrix algorithms [14, 26, 28, 18, 15].

The emphasis of this work is to present an intuitive and readily implementable algorithm to compute the low-rank SVD through sketching [22]. The presented algorithm takes full advantage of highly sparsified random test matrices. Further, the formulation of the algorithm provides some interesting flexibility to update the prior on the target-rank based on the approximated singular value spectrum, before recovering both the left and right singular vectors. Thus the expensive computational steps are performed after cheaply sketching the singular value spectrum.

In the following, we provide context for our cSVD algorithm, examining different measurement matrices and comparing the performance with the randomized SVD. The organization is as follows: First, Section 2 briefly reviews the SVD and randomized algorithms as well as the basic concept of sparsity and low-rank structure. Then, the compressed SVD algorithm is described in Section 3. The results are presented in Section 4. Final remarks are given in Section 5.

2. Background

2.1. Singular Value Decomposition

Given a real matrix $\mathbf{X}$ of dimension $m \times n$, the singular value decomposition admits the following factorization

$$\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^\top, \tag{1}$$

where the left and right singular vectors are denoted as $\mathbf{U}$ and $\mathbf{V}$, respectively. The spectrum of the data is described by the singular values, which are the diagonal elements of the matrix $\mathbf{S}$. Instead of the full factorization, it is often of interest to compute a low-rank matrix approximation. Specifically, the optimal k-rank approximation $\mathbf{X}_k$ to the matrix $\mathbf{X}$ in the least square sense that minimizes the Frobenius or 2-norm is given by

$$\mathbf{X}_k := \mathbf{U}_k\mathbf{S}_k\mathbf{V}_k^\top := \underset{\mathbf{X}'_k}{\operatorname{argmin}} \|\mathbf{X} - \mathbf{X}'_k\|. \tag{2}$$

The low-rank approximation can be obtained by simply truncating the singular values and vectors computed via a deterministic algorithm [16]. However, this approach becomes infeasible for high-dimensional data matrices. Instead, iterative algorithms like Krylov-subspace methods can be used to approximate the dominant singular vectors more efficiently [16]. These partial algorithms are the most popular approach for computing low-rank matrix approximations, in particular, for large sparse matrices. However, randomized algorithms have become increasingly popular over the last two decades. This is because randomized algorithms often show a better performance in practice and are more robust than Krylov methods [25].

2.2. Randomized Low-Rank Approximations

In 1998, Frieze et al. [14] presented a rigorous approach to efficiently compute the approximate low-rank SVD using a Monte Carlo based approach. They showed that a matrix can be approximated from a much smaller sub-matrix obtained through non-uniform row and column sampling. The rows and columns are selected by importance sampling based on leverage scores. Further improvements and theoretical results have since been obtained [19, 8, 2].

Sarlos [28] and Martinsson et al. [26] introduced an approach based on random projections. They use the properties of random vectors to efficiently build a subspace that captures the range of the matrix. This approach was further improved by Woolfe et al. [32], and eventually Halko et al. [18] unified and extended this work in their seminal paper. The *randomized singular value decomposition* (rSVD) is now considered a state-of-the-art algorithm for computing a low-rank matrix approximation [24]. For implementation details see [30, 13].

A third line of work based on compressive measurements is closely related to matrix sketching [22, 31]. Gilbert et al. [15] presented a single-pass algorithm, and corresponding error bounds, to compute the singular values and right singular vectors from a sketched matrix for a streaming data model. The various approaches outlined here highlight the growing importance of randomized techniques for matrix decompositions.

2.3. Sparsity and Low-Rank Structure

The theory of compressed sensing [6, 9] demonstrates that a signal can be approximately reconstructed from a low-dimensional subsample using fewer measurements than required by the Shannon-Nyquist sampling theorem. The key assumption is that the data is compressible i.e., the signal is sparse in some domain. This is certainly the case for highly structured signals like audio and images. Consider a compressible signal $\mathbf{x} \in \mathbb{R}^m$ which is sparse with respect to some transform basis $\mathbf{\Psi}$, so that $\mathbf{x} = \mathbf{\Psi}\mathbf{s}$. Then we can formulate the following model to obtain a compressed signal $\mathbf{y} \in \mathbb{R}^s$ as

$$\mathbf{y} := \mathbf{\Phi}\mathbf{x} = \mathbf{\Phi}\mathbf{\Psi}\mathbf{s}, \tag{3}$$

where $\mathbf{\Phi} \in \mathbb{R}^{s \times m}$ is a random test matrix, also called measurement matrix. Specifically, $\mathbf{\Phi}$ is a linear map $\mathbb{R}^m \mapsto \mathbb{R}^s$, reducing the dimensionality, while preserving the information. We do not need explicit knowledge about the transform basis $\mathbf{\Psi}$ in order to obtain $\mathbf{y}$. Instead, we can understand $\mathcal{I} := \mathbf{\Phi}\mathbf{\Psi}$ as an information operator, sampling s pieces of information about the underlying sparse signal $\mathbf{s}$ [9].

More generally, we are often interested in highly structured 2-D or 3-D signals, such as image or video data. Signals of this type are stored as arrays and feature a natural low-rank structure. Hence, we want to generalize the model presented in Eq. (3) to

$$\mathbf{Y} := \mathbf{\Phi}\mathbf{X}, \tag{4}$$

where we refer to $\mathbf{X} \in \mathbb{R}^{m\times n}$ as the high-dimensional data and $\mathbf{Y} \in \mathbb{R}^{s\times n}$ as the compressed data or sketch of the data.

3. Compressed Singular Value Decomposition

Despite increasing computational power, big data matrices pose a tremendous computational challenge for current deterministic SVD algorithms. In such cases an approximate SVD is often the only feasible computation and may be suitable for many uses. The only disadvantage comes from incurring an error penalty $\epsilon \geq 0$ so that the approximate low-rank SVD satisfies

$$\|\mathbf{X} - \hat{\mathbf{X}}_k\|_F \leq \|\mathbf{X} - \mathbf{X}_k\|_F \cdot (1+\epsilon).$$

Ideas from compressed sensing provide a computationally efficient framework to approximate the SVD from just a few incoherent measurements. Therefore, we assume that the data are low dimensional, which implies that the columns are sparse in some transform basis $\mathbf{\Psi}$ so that $\mathbf{X} = \mathbf{\Psi}\mathbf{S}$.

The concept of matrix compression enables one to capture and represent the information of data matrix $\mathbf{X}$ in a much smaller compressed matrix $\mathbf{Y}$. If we understand each vector in our data matrix as a data point in a high-dimensional space, then from an information retrieval perspective we are mainly interested in the geometric relationships between the vectors, so that similarities and differences can be identified [3]. The compressed matrix must ensure that distances and angles between the data points are preserved. The restricted isometry property (RIP) guarantees this property with high probability, when using a sufficient number of incoherent measurements [5]. Thus, compression can be understood as a sampling process that captures a parsimonious representation of the dominant information of the data matrix. Interestingly, many high-dimensional signals arising in image and video processing applications have a low intrinsic rank relative to the dimension of the ambient measurement space. Specifically, the theory of compressed sensing demonstrates that certain data can be well-approximated from massively under-sampled or compressed representations. However, depending on the underlying structure of the data, the compressed SVD may require many fewer measurements than the compressed sensing theory suggests. This is because we actually have access to the full high-dimensional data

3.1. Compressed Algorithm

In order to obtain a low-rank SVD approximation of $\mathbf{X} \in \mathbb{R}^{m\times n}$ with target-rank k and $m \geq n$, we require a random test matrix $\mathbf{\Phi} \in \mathbb{R}^{l\times m}$, where $l = k + p$. Here p denotes a parameter for oversampling, and we shall see that a small amount of oversampling is often sufficient in practice. From Eq. (4) we obtain the compressed data matrix $\mathbf{Y} \in \mathbb{R}^{l\times n}$ as

$$\mathbf{Y} := \mathbf{\Phi}\mathbf{X}.$$

Next, our aim is it to approximate the right singular vectors. Recall, that the singular values and vectors can be related to the eigendecomposition of the inner and outer dot product of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ as

$$\mathbf{A}^\top\mathbf{A} = (\mathbf{VSU}^\top)(\mathbf{USV}^\top) = \mathbf{VDV}^\top, \quad (5)$$
$$\mathbf{A}\mathbf{A}^\top = (\mathbf{USV}^\top)(\mathbf{VSU}^\top) = \mathbf{UDU}^\top, \quad (6)$$

where the eigenvalues are the squared singular values, i.e., $\mathbf{D} = \mathbf{S^2}$. Thus, we proceed by forming a smaller matrix $\mathbf{B} \in \mathbb{R}^{l\times l}$ first

$$\mathbf{B} := \mathbf{YY}^\top, \quad (7)$$

followed by computing the eigendecomposition

$$\mathbf{B} =: \mathbf{T}\tilde{\mathbf{D}}\mathbf{T}^\top. \quad (8)$$

The matrix $\mathbf{T} \in \mathbb{R}^{l\times l}$ and $\tilde{\mathbf{D}} \in \mathbb{R}^{l\times l}$ are the approximate eigenvectors and eigenvalues, respectively. Hence, the approximate singular values are $\tilde{\mathbf{S}} = \sqrt{\mathbf{D}}$. If, $l > k$ we truncate the decomposition, i.e., we extract the first k dominant eigenvectors *trunc*$(\mathbf{T}, k)$ and singular values *trunc*$(\tilde{\mathbf{S}}, k)$. At this point of the algorithm we can also update our prior decision of the target-rank k based on the approximated singular value spectrum, i.e., a smaller target-rank might be sufficient. This flexibility of the algorithm might be interesting in several applications like robust principal component analysis.

From Equation (1) it follows that either the left or right singular vectors can be approximated from each other as $\mathbf{U} = \mathbf{XVS}^{-1}$ or $\mathbf{V} = \mathbf{X}^\top\mathbf{US}^{-1}$. Thus, the approximate right singular vectors $\tilde{\mathbf{V}} \in \mathbb{R}^{n\times k}$ are recovered as

$$\tilde{\mathbf{V}} := \mathbf{Y}^\top\mathbf{T}\tilde{\mathbf{S}}^{-\mathbf{1}}. \quad (9)$$

Similarly, the approximate left singular vectors $\tilde{\mathbf{U}} \in \mathbb{R}^{m\times k}$ are obtained as follows

$$\tilde{\mathbf{U}} := \mathbf{X}\tilde{\mathbf{V}}\tilde{\mathbf{S}}^{-\mathbf{1}}. \quad (10)$$

This last step involves a second pass over the high-dimensional input matrix. However, the problem in practice is that the columns of $\tilde{\mathbf{U}}$ are only approximately orthogonal. This is due to the approximation errors in the eigenvalues. Thus, we need to introduce an additional updating step. Specifically, we compute the SVD of the scaled right singular vectors (principal components)

$$\tilde{\mathbf{U}}\tilde{\mathbf{S}} =: \mathbf{USQ}^\top. \quad (11)$$

Then, the last step is to update of the right singular vectors

$$\mathbf{V} := \tilde{\mathbf{V}}\mathbf{Q}, \quad (12)$$

which completes the approximate computation of the singular value decomposition. A justification for this approach is motivated as follows

$$\left[\tilde{\mathbf{U}}\tilde{\mathbf{S}}\right]\tilde{\mathbf{V}}^\top =: \left[\mathbf{USQ}^\top\right]\hat{\mathbf{V}}^\top =: \mathbf{USV}^\top \approx \mathbf{X}_k. \quad (13)$$

The computational steps are summarized in Algorithm 1. An alternative (numerically more stable) formulation is outlined in Algorithm 2 in Appendix A.

Algorithm 1 Compressed SVD (cSVD)

Input: Input matrix $\mathbf{X}$ of dimension $m \times n$, and target rank k.
Optional: Parameters p to control oversampling.

```
(1)  $l \leftarrow k + p$                                             Slight oversampling
(2)  $\mathbf{\Phi} \leftarrow \texttt{rand}(l, m)$                    Generate $l \times m$ random test matrix.
(3)  $\mathbf{Y} \leftarrow \mathbf{\Phi} * \mathbf{X}$                 Sketch input matrix.
(4)  $\mathbf{B} \leftarrow \mathbf{Y} * \mathbf{Y}^\top$               Form smaller $l \times l$ matrix.
(5)  $\mathbf{B} \leftarrow \frac{1}{2} \cdot (\mathbf{B} + \mathbf{B}^\top)$   Ensure symmetry.
(6)  $\mathbf{T}, \mathbf{D} \leftarrow \texttt{eig}(\mathbf{B}, k)$   Truncated eigendecomposition.
(7)  $\tilde{\mathbf{S}} \leftarrow \sqrt{\mathbf{D}}$                  Rescale eigenvalues.
(8)  $\tilde{\mathbf{V}} \leftarrow \mathbf{Y}^\top * \mathbf{T} * \tilde{\mathbf{S}}^{-1}$   Approx. right singular values.
(9)  $\tilde{\mathbf{U}} \leftarrow \mathbf{X} * \tilde{\mathbf{V}}$    Approx. unscaled left singular vals.
(10) $\mathbf{U}, \mathbf{S}, \mathbf{Q}^\top \leftarrow \texttt{svd}(\tilde{\mathbf{U}})$   Update left singular vectors and vals.
(11) $\mathbf{V} \leftarrow \tilde{\mathbf{V}}\mathbf{Q}$               Update right singular vectors.
```

Return: $\mathbf{U} \in \mathbb{R}^{m \times k}$, $\mathbf{S} \in \mathbb{R}^{k \times k}$ and $\mathbf{V} \in \mathbb{R}^{n \times k}$

Remark 1. If the input matrix has low-rank structure, a small amount of oversampling is sufficient, i.e., $p = \{10, 20\}$.

Remark 2. The computational performance depends considerably on the random test matrix used for sketching in Step 3. Very sparse or single pixel measurements (uniform random selected rows), as discussed in Section 3.3, perform astonishingly well in image and video processing applications.

Remark 3. It is optional to compute power (subspace) iterations [30] after Step 3 to improve the quality of the sketch.

Remark 4. Numerical issues can arise, if k is chosen larger than the actual intrinsic rank r of the input matrix. Specifically, in this case the eigenvalues $\{\lambda_i\}_{i>r}$ are close to zero or negative, which can lead to an arithmetic underflow in Step 7 and 8. Thus, these eigenvalues and the corresponding eigenvectors are masked.

3.2. Computational Complexity Analysis

To contextualize the computational complexity of Algorithm 1 in terms of the number of floating points operations, we evaluate the complexity of each step:

Step 1: Requires $\mathcal{O}(1)$ operations.
Step 2: Generating a $l \times m$ (sparse) random test matrix requires $\mathcal{O}(\mathbf{nnz}(\Phi))$ operations, where $\mathbf{nnz}(\cdot)$ is the number of nonzero entries.
Step 3: Forming the sketch $\mathbf{Y}$ requires $\mathcal{C}_{\mathbf{\Phi}} \cdot n$ operations, where $\mathcal{C}_{\mathbf{\Phi}}$ denotes the complexity of applying the random test matrix to an $m \times 1$ column vector of $\mathbf{X}$. If the random test matrix is dense, then $\mathcal{C}_{\mathbf{\Phi}} = \mathcal{O}(lm)$.
Step 4: Computing $\mathbf{B}$ requires $\mathcal{O}(l^2 n)$ operations.
Step 5: Ensuring symmetry requires $\mathcal{O}(l^2)$ operations.
Step 6: The eigendecomposition requires $\mathcal{O}(l^3)$.
Step 7: Requires $\mathcal{O}(k)$ operations.
Step 8: Computing $\tilde{\mathbf{V}}$ requires $\mathcal{O}(nlk + lk)$ operations.
Step 9: Computing $\tilde{\mathbf{U}}$ requires $\mathcal{O}(mnk)$ operations.
Step 10: Computing the SVD requires $\mathcal{O}(mk^2)$ operations.
Step 11: Updating $\mathbf{V}$ requires $\mathcal{O}(nk^2)$ operations.

In summary, the dominant complexities sum up to

$$C_{cSVD} = \mathcal{C}_{\mathbf{\Phi}} \cdot n + \mathcal{O}(l^2 n + l^3 + nlk + mnk + mk^2) \quad (14)$$

operations, and the algorithm is dominated by the complexity $\mathcal{O}(mnk)$. However, steps (2), (3), (4), (5), (8), (9) and (11) are embarrassingly parallel and can substantially benefit from GPU accelerated computations. If the data matrix does not fit into fast memory, the algorithm's bottleneck is posed by the required number of sequential reads of the entire data matrix. The transfer of data from slow to fast memory is relatively expensive and can dominate the actual flop count of the algorithm. Like the randomized SVD algorithm, the proposed compressed algorithms require only two passes over the data matrix $\mathbf{X}$. If the input matrix is to big to be accessed a second time, single pass algorithms are an interesting alternative [29].

3.3. Measurement Matrices

To construct an efficient measurement matrix we rely on randomness. The Gaussian random test matrix is the most prominent choice, consisting of independent identically distributed (iid) standard normal entries. It follows that columns or rows are linear independent with high probability. Yet, Gaussian measurement matrices have drawbacks: they are expensive to generate and dense matrix multiplications are computationally intensive. This can make the method infeasible when dealing with high-dimensional data. However, there are several interesting alternatives for sketching a matrix.

Matrices in image and video processing applications often feature some natural structure, i.e., the mass is well distributed and the matrix is sparse in some domain. In this case it is sufficient to construct even simpler sensing matrices. A most memory efficient sensing matrix, which also makes the matrix multiplication for compression redundant is to use 'single-pixel' (spixel) measurements, i.e., uniform random row sampling [7]. Specifically, the sketch $\mathbf{Y}$ is constructed by choosing l random rows without replacement from $\mathbf{X}$. In addition, the signs of each selected row can be randomly flipped to introduce additional randomness into the sampling process. This measurement matrix is also known as the 'CountSkech' [31]. However, this approach is prone to fail if the data mass is unevenly centered.

Another highly efficient approach is based on sparse random projections, originally introduced by Achlioptas [1]. Specifically, a sparse sensing matrix $\mathbf{\Phi}$ with only a few nonzero iid random entries is constructed (see Appendix B). Li et al. [21] demonstrated that accurate results can be achieved even with more aggressive (highly sparse) random test matrices. Hence, by leveraging high-performance sparse matrix multiplication routines, the sketch $\mathbf{Y}$ can be obtained in a fraction of the time as compared to using a dense random test matrix.

4. Experimental Evaluation

In the following we evaluate the proposed cSVD algorithm, and compare its performance to the randomized SVD. Both algorithms are implemented in *Python* as part of the open-software package *ristretto* (GIT repository: https://github.com/Benli11/ristretto). All computations were performed on a machine with the following specifications: Intel Core i7-6500U CPU (2.5GHz), and 16GB DDR3 memory. The relative reconstruction error is computed as $||\mathbf{X} - \mathbf{X}_k||_F / ||\mathbf{X}||_F$, where the rank k approximation is denoted as $\mathbf{A}_k$.

4.1. Image Compression

The singular value decomposition is a standard tool for image and video compression. A natural image can be reconstructed in general from a very small subset of dominant singular values and vectors. Here we use Canaletto's famous painting 'Bucentaur's return to the pier by the Palazzo Ducale', which is provided by the Google Art Project as a high resolution image with 30000×20857 pixels.[1]

First, we use a lower resolution to 4096×2848 pixels. For computational convince we stack the three color channels together so that we yield an array of dimension 12288×2848. Then we seek to reveal the dominant structure of the image by computing the first $k = 500$ singular values and vectors. Figure 3 shows the original image as well as the low-rank approximation computed via the randomized and compressed SVD algorithms, using a small amount of oversampling $p = 10$. By visual inspection the reconstruction error is hardly noticeable; however, the randomized SVD algorithm achieves a slightly lower relative error as shown in Table 1. This is because the randomized SVD approximates the column space of the input matrix, thus more information is used. Another advantage of the randomized SVD is that the approximation quality can be considerably improved by computing additional power iterations. Performing two power iterations $q = 2$ achieves a near optimal reconstruction error compared to the deterministic (truncated) SVD. Nevertheless, this comes with higher computational costs. The approximation quality of the compressed SVD might be sufficient in a variety of applications, with the clear advantage of being computationally cheap, especially with increasing dimensions.

	Time (s)	Speedup	Error
Truncated SVD	39.46	*	0.083
rSVD (p=10, q=0)	2.4	16.4	0.111
rSVD (p=10, q=1)	4.49	8.8	0.088
rSVD (p=10, q=2)	6.72	5.9	0.085
cSVD sparse (p=10)	2.3	17.1	0.111
cSVD spixel (p=10)	1.7	23.2	0.112

Table 1: Computational results for Canaletto's low-resolution painting (12288×2848). The * indicates baseline.

Next, we compute the $k = 500$ dominant singular values and vectors on the grayscale high resolution image. The results are summarized in Table 2. The 30000×20857 array representing the image is to big to be decomposed by the deterministic SVD algorithm on our test workstation, i.e., the machine runs out of memory. Both the randomized and compressed SVD algorithm are able to provide the low-rank approximation with ease. However, it can be seen that the computation of power iterations requires a significant amount of additional computational resources on an array of this dimension. The compressed SVD using single-pixel measurements achieves higher speedups while attaining competitive reconstruction errors.

	Time (s)	Speedup	Error
Truncated SVD	-	-	-
rSVD (p=10, q=0)	25.8	3.2	0.132
rSVD (p=10, q=1)	52.93	1.5	0.110
rSVD (p=10, q=2)	82.01	*	0.108
cSVD sparse (p=10)	19.01	4.3	0.132
cSVD spixel (p=10)	13.1	6.7	0.132

Table 2: Computational results for the high-resolution grayscale image (30000×20857). Truncated SVD failed due to limited memory. The * indicates baseline.

The dominant singular values are shown in Figure 2. All algorithms capture faithfully the dominant spectral information. However, the approximated singular values by both the

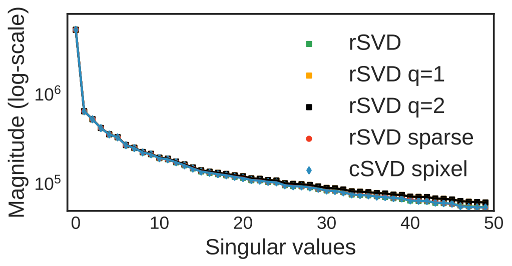
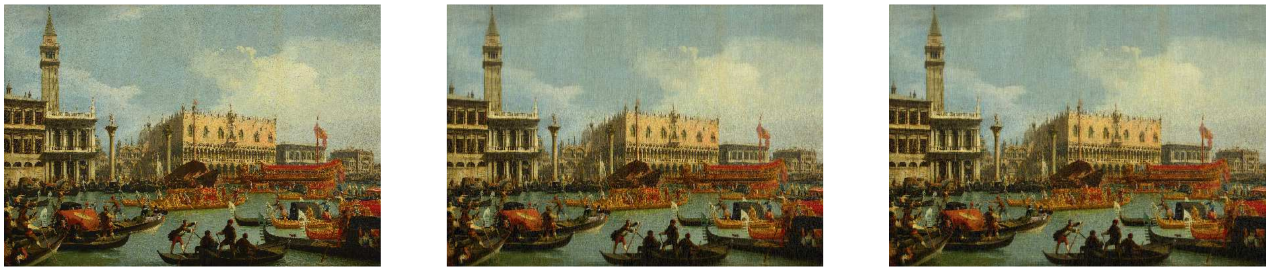


Figure 2: Singular values of Canaletto's painting approximated by the randomized and compressed algorithm.

[1] Canaletto's painting shows a lively scene of the pier on Ascension day in Venice. Since 1930 it has been part of the collection of the Pushkin State Museum of Fine Arts, Moscow. See google's art project for more details: https://www.google.com/culturalinstitute/beta/asset/mwEV7sO9uSFCpw.



(a) Input image. (b) Reconstructed using rSVD. (c) Reconstructed using cSVD (sparse).

Figure 3: Canaletto's famous painting 'Bucentaur's return to the pier by the Palazzo Ducale' is used to demonstrate the image compression performance of both the rSVD and cSVD algorithm. By visual inspection no distinct difference is notable.

compressed and randomized algorithm (without power iterations) start to fall of for $k > 20$. The performance could be slightly improved by increasing the amount of oversampling.

4.2. Background/Foreground Separation

Background/foreground separation is an integral task in video surveillance; however, estimating a good model for the background is computationally challenging. Robust principal component analysis (RPCA) is a viable candidate for this task. The idea is to formulate the problem as a matrix separation problem. Specifically, it is assumed that the background is approximately low-rank, while foreground objects are treated as sparse errors. Bouwmans et al. [4] provide a comprehensive survey of matrix separation techniques suitable for background/foreground separation. Despite the outstanding performance of many of these techniques, the computational costs are tremendous for high-resolution video streams. This is because the computationally expensive singular value decomposition is the work-horse algorithm behind many of these techniques. Erichson et al. [13] have proposed using the randomized SVD to ease the computational demands, and Oh et al. [27] have proposed a fast randomized singular value thresholding algorithms for low-rank optimization. Similarly we evaluate and compare the compressed SVD for this task. Specifically, we use the method of inexact augmented Lagrange multipliers (IALM) [23] to obtain the background/foreground separation.

The results for separating 200 frames of a static surveillance video of resolution 480×720 are shown in Figure 4 and Table 3. Here, the 'snowFall' video sequence of the CDNET database were used [17]. On first glance the results might seem surprising. The compressed SVD algorithm using single-pixel measurements is not only substantially faster, but also achieves a better performance in terms of the F-measure. This phenomena is explained by the intrinsic regularization effect of randomized methods [24]. In particular, sampling random rows improves the accuracy of the algorithm, which is similar to the concept of bagging in statistics. It is interesting that the weaker performance in terms of the reconstruction error can improve the accuracy in terms of the F-measure. Further, we see that the compressed SVD algorithm is computational more efficient for tall and skinny matrices like video sequences. The advantage becomes pronounced with increasing dimension as demonstrated in the next section. The IALM algorithm could be further improved by taking advantage of the compressed algorithm's ability to threshold the number of left and right singular vectors before actually computing them.

	Time (s)	**Speedup**	**F-measure**
Truncated SVD	258.25	*	0.663
rSVD (p=10, q=0)	141.99	1.8	0.705
cSVD sparse (p=10)	115.38	2.2	0.712
cSVD spixel (p=10)	113.64	2.3	0.749

Table 3: Computational results of the IALM method using different SVD algorithms. The cSVD algorithm using single-pixel measurements shows the best performance.

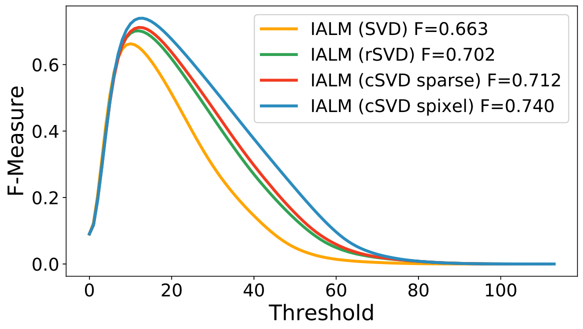
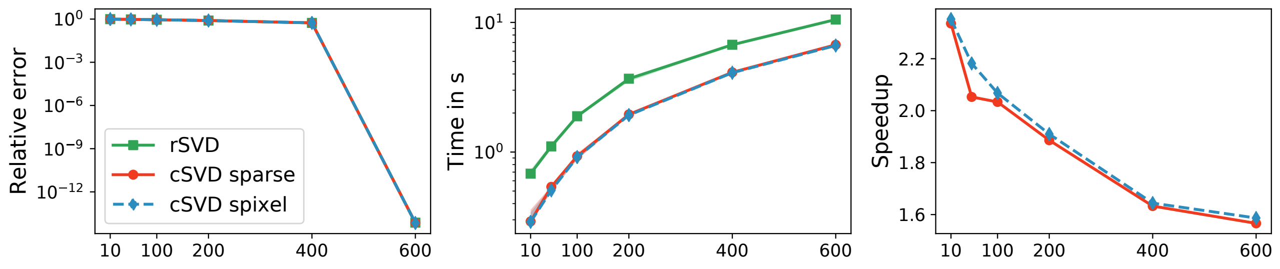
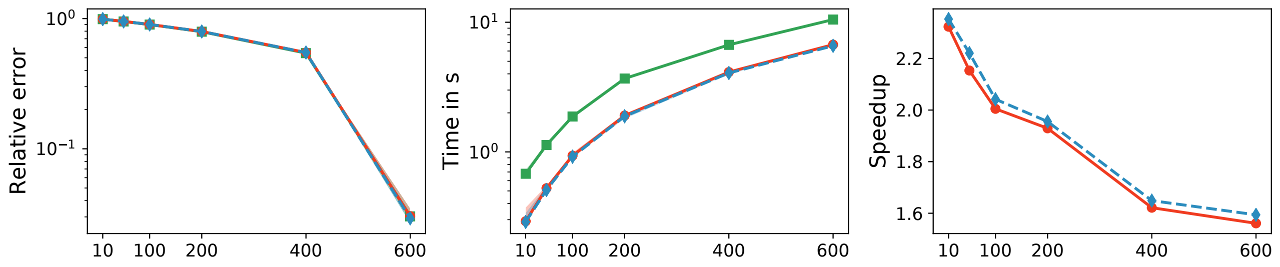


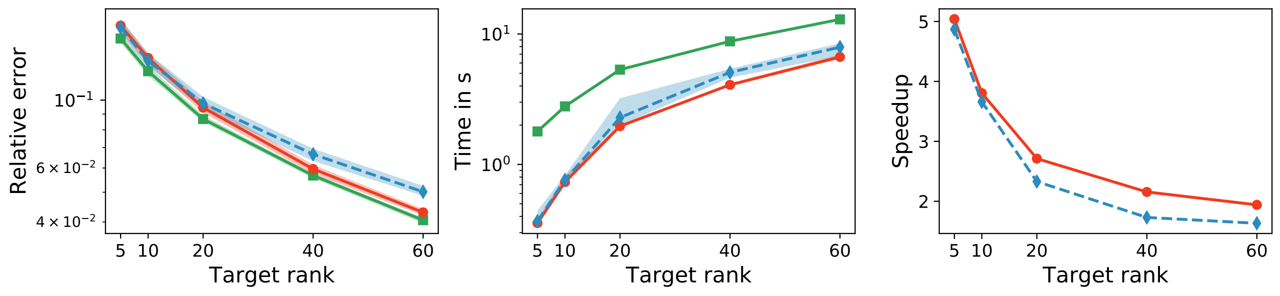
Figure 4: Performance evaluation of the IALM method using the deterministic SVD algorithm as well as the randomized and compressed algorithm. Interestingly, cSVD with single-pixel measurements outperforms the other methods.



(a) Synthetic 20000×10000 matrix of rank $r = 600$.



(b) Synthetic 20000×10000 matrix of rank $r = 600$, perturbed with 10% white noise.



(c) High definition 1080×1920 video sequence with 200 frames.

Figure 5: Algorithm runtime, speedups and relative reconstruction errors for varying target ranks. The compressed algorithm shows substantial speedups, in particular, for computing the low-rank approximation for the high definition video sequence (tall and skinny matrix). Here the speedups of the cSVD are computed using the rSVD as baseline algorithm.

4.3. Timing

To contextualize the computational performance we evaluate the compressed algorithm on synthetic low-rank matrices and a high definition video sequence. The randomized SVD (without power iterations) is used as benchmark. Both methods require two passes over the input matrix, and a slight oversampling parameter of $p = 10$ is used. It is important to note that the computational performance of the rSVD could be increased by using random structured or very sparse test matrices.

Figure 5a shows the algorithm runtime, speedups and relative reconstruction errors averaged over 20 runs for a synthetic 20000×10000 matrix of rank $r = 600$. The compressed SVD using both sparse (labeled 'sparse') and single-pixel (labeled 'spixel') measurements achieves speedups over the randomized SVD, while attaining nearly the same reconstruction errors. The experiment is repeated in the presence of white noise in Figure 5b, and the results appear to be very similar. Further, we note that both methods are somewhat robust to noise.

The performance results for the high definition video sequence are more interesting. Specifically, we have computed various low-rank approximations for the tall and skinny matrix formed by stacking 200 frames with resolution of 1080×1920 pixels. Here, the compressed SVD algorithm shows an improvement in computational time over the randomized SVD by a factor of about 2 to 5 across the varying target ranks. Interestingly, high-dimensional data can often be massively under-sampled while still preserving the dominant structure. Thus, the reconstruction error is competitive despite the enormous compression factor as demonstrated here. Overall, sparse random measurements achieve a better reconstruction error, while exhibiting less variability than single-pixel measurements (uniform random selected rows). This is because the sparse random test matrix has the ability to capture slightly more information across the data matrix.

5. Conclusion

Massive data poses a tremendous computational challenge for deterministic SVD algorithms, despite modern computer power. Indeed, data acquisition is far outstripping computational resources, necessitating algorithms that scale with the underlying signal complexity rather than the size of the ambient measurement space. In computer vision, for instance, there is a growing demand for higher-resolution imaging, exemplified by 4K video, which produces approximately 24 million data points per image in RGB. Hence, algorithms based on compressed representations of the data present a promising and enabling alternative for computing matrix factorizations in this domain; some advances include randomized, sketched, and Monte Carlo methods, in addition to the cSVD presented here.

The randomized SVD as formulated by [26, 18] is the best off-the-shelf randomized algorithm for computing low-rank matrix approximations. The algorithm is robust, reliable and the approximation quality can be controlled via oversampling and power iterations. Further, the algorithm is mathematical sound and comes with strong error bounds. Still, there is room for innovations and modifications.

Inspired by the compressed sensing literature, we demonstrate a computationally efficient two-pass algorithm to compute the approximate low-rank SVD of a large data matrix from a small compressed matrix. Despite the limited difference to previous proposed randomized algorithms, the subtle modifications compare favorably with the randomized SVD in image and video processing applications. Further, the ideas can also be used to compute the dynamic mode decomposition [11, 12]. The cSVD algorithm provides a interesting trade-off between precision and speed in these specific applications. This is mainly achieved by exploiting aggressive sampling strategies, i.e., single-pixel measurements and very sparse random test matrices. The results show that the cSVD algorithm is highly suitable for the task of approximating large imagery like data as well as to approximate large tall and skinny matrices. Thereby, the computational advantage becomes pronounced with increasing dimensions. The drawback, however, is that the approximation quality can not be further improved via power iterations as the randomized SVD algorithm does.

Future research focuses on better utilizing the additional flexibility provided by the compressed algorithm. Specifically, it is interesting to update the prior on the target-rank based on the sketched singular value spectrum. This is of interest in applications like RPCA, where algorithms employ a hard-threshold based on the singular value spectrum. Thus, the computational costs can be reduced by computing only the relevant left and right singular vectors. Further, as with other randomized algorithms, cSVD is readily parallelized and can benefit from a GPU accelerated implementation.

A. Compressed Algorithm II

Algorithm 2 presents a different implementation of the compressed SVD algorithm, which is slightly more computationally expensive. However, the accuracy and numerical stability of the algorithm is improved by computing the deterministic SVD of $\mathbf{Y}$ in step 4, instead of using the eigendecomposition of the smaller matrix $\mathbf{B}$. Thus, in practice we favor this algorithm, in particular, for tall and skinny matrices. However, for fat matrices which have a large dimension n it becomes more efficient to form the smaller matrix $\mathbf{B}$ first, as outlined in Algorithm 1.

Algorithm 2 Compressed SVD (cSVD) version II

```
Input: Input matrix $\mathbf{X}$ of dimension $m \times n$, and target rank $k$.
Optional: Parameters $p$ to control oversampling.
(1) $l \leftarrow k + p$                                              Slight oversampling
(2) $\mathbf{\Phi} \leftarrow \texttt{rand}(l, m)$                    Generate $l \times m$ random matrix.
(3) $\mathbf{Y} \leftarrow \mathbf{\Phi} * \mathbf{X}$                Sketch input matrix.
(4) Optional: compute power (subspace) iterations [30].
(5) $\mathbf{T}, \tilde{\mathbf{S}}, \tilde{\mathbf{V}} \leftarrow \texttt{svd}(\mathbf{Y}, k)$     Compute truncated SVD.
(6) $\mathbf{U}, \mathbf{S}, \mathbf{Q}^\top \leftarrow \texttt{svd}(\mathbf{X} * \tilde{\mathbf{V}})$   Update decomposition.
(7) $\mathbf{V} \leftarrow \tilde{\mathbf{V}}\mathbf{Q}$              Update right singular vectors.
Return: $\mathbf{U} \in \mathbb{R}^{m \times k}$, $\mathbf{S} \in \mathbb{R}^{k \times k}$ and $\mathbf{V} \in \mathbb{R}^{n \times k}$
```

B. Sparse Random Test Matrix

Following the ideas of Achlioptas [1], a sparse random test matrix $\mathbf{\Phi}$ can be constructed by drawing entries ϕ_{ij} from the following distribution

$$\phi_{ij} = \sqrt{c} \begin{cases} 1 & \text{with prob. } \frac{1}{2c} \\ 0 & \text{with prob. } 1 - \frac{1}{c} \\ \text{-1} & \text{with prob. } \frac{1}{2c} \end{cases} \tag{15}$$

The parameter c controls the density of the nonzero entries, e.g. $c = 2, 3$. Li et al. [21] demonstrated that accurate results can be achieved even with more aggressive (highly sparse) sampling rates like $c = \sqrt{m}$ or $c = m/log(m)$. In practice, we obtained that replacing the nonnegative entries by uniform distributed random entries performs slightly better. Further, it is interesting to note that it is very efficient to premultiply a dense matrix by a sparse matrix using the compressed sparse row (CSR) matrix multiplication routines in *SciPy* [20]. Post multiplying a dense matrix by a sparse matrix is not as efficient. It is also not efficient to premultiply a in place transposed matrix. Thus, the randomized SVD does not benefit as much as the compressed SVD from sparse random test matrices, if implemented in *Python*.

References

[1] D. Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4):671–687, 2003. 4, 8

[2] D. Achlioptas and F. McSherry. Fast computation of low-rank matrix approximations. *Journal of the ACM*, 54(2):9, 2007. 2

[3] M. W. Berry, Z. Drmac, and E. R. Jessup. Matrices, vector spaces, and information retrieval. *SIAM Review*, 41(2):335–362, 1999. 3

[4] T. Bouwmans, A. Sobral, S. Javed, S. K. Jung, and E.-H. Zahzah. Decomposition into low-rank plus additive matrices for background/foreground separation: A review for a comparative evaluation with a large-scale dataset. *Computer Science Review*, 23:1–71, 2017. 6

[5] E. J. Candès, J. K. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics*, 59(8):1207–1223, 2006. 3

[6] E. J. Candès and M. B. Wakin. An Introduction to Compressive Sampling. *IEEE Signal Processing Magazine*, 25(2):21–30, 2008. 2

[7] M. B. Cohen, Y. T. Lee, C. Musco, C. Musco, R. Peng, and A. Sidford. Uniform sampling for matrix approximation. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 181–190. ACM, 2015. 4

[8] A. Deshpande and S. Vempala. Adaptive sampling and fast low-rank matrix approximation. In *Proceedings of the 10th International Conference on Randomization and Computation*, pages 292–303, 2006. 2

[9] D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006. 2

[10] E. Drinea, P. Drineas, and P. Huggins. A randomized singular value decomposition algorithm for image processing applications. In *In Proceedings of the 8th panhellenic conference on informatics*, pages 278–288, 2001. 1

[11] N. B. Erichson, S. L. Brunton, and J. N. Kutz. Compressed dynamic mode decomposition for background modeling. *Journal of Real-Time Image Processing*, Nov 2016. 8

[12] N. B. Erichson, S. L. Brunton, and J. N. Kutz. Randomized dynamic mode decomposition. *arXiv preprint arXiv:1702.02912*, 2017. 8

[13] N. B. Erichson, S. Voronin, S. L. Brunton, and J. N. Kutz. Randomized matrix decompositions using R. *arXiv preprint arXiv:1608.02148*, 2016. 2, 6

[14] A. Frieze, R. Kannan, and S. Vempala. Fast Monte-Carlo algorithms for finding low-rank approximations. *Journal of the ACM*, 51(6):1025–1041, 2004. 1, 2

[15] A. C. Gilbert, J. Y. Park, and M. B. Wakin. Sketched SVD: Recovering Spectral Features from Compressive Measurements. *arXiv preprint arXiv:1211.0361*, pages 1–10, 2012. 1, 2

[16] G. H. Golub and C. F. Van Loan. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996. 2

[17] N. Goyette, P. M. Jodoin, F. Porikli, J. Konrad, and P. Ishwar. Changedetection.net: A new change detection benchmark dataset. In *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8, June 2012. 6

[18] N. Halko, P.-G. Martinsson, and J. A. Tropp. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM Review*, 53(2):217–288, 2011. 1, 2, 8

[19] S. Har-Peled. Low rank matrix approximation in linear time. *arXiv preprint arXiv:1410.8802*, pages 1–10, 2014. 2

[20] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. 8

[21] P. Li, T. J. Hastie, and K. W. Church. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 287–296. ACM, 2006. 4, 8

[22] E. Liberty. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 581–588. ACM, 2013. 2

[23] Z. Lin, M. Chen, and Y. Ma. The augmented lagrange multiplier method for exact recovery of corrupted low-rank matrices. *arXiv preprint arXiv:1009.5055*, 2010. 6

[24] M. W. Mahoney. Randomized Algorithms for Matrices and Data. *Foundations and Trends in Machine Learning*, 3(2):123–224, 2011. 1, 2, 6

[25] P.-G. Martinsson. Randomized methods for matrix computations and analysis of high dimensional data. *Preprint arXiv:1607.01649*, pages 1–55, 2016. 2

[26] P.-G. Martinsson, V. Rokhlin, and M. Tygert. A randomized algorithm for the decomposition of matrices. *Applied and Computational Harmonic Analysis*, 30(1):47–68, 2011. 1, 2, 8

[27] T. H. Oh, Y. Matsushita, Y. W. Tai, and I. S. Kweon. Fast randomized singular value thresholding for low-rank optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP(99):1–1, 2017. 6

[28] T. Sarlos. Improved approximation algorithms for large matrices via random projections. In *Foundations of Computer Science. 47th Annual IEEE Symposium on*, pages 143–152, 2006. 1, 2

[29] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher. Randomized single-view algorithms for low-rank matrix approximation. *arXiv preprint arXiv:1609.00048*, 2016. 4

[30] S. Voronin and P.-G. Martinsson. RSVDPACK: Subroutines for computing partial singular value decompositions via randomized sampling on single core, multi core, and GPU architectures. *arXiv preprint arXiv:1502.05366*, pages 1–15, 2015. 2, 4, 8

[31] D. P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends in Theoretical Computer Science*, 10(1-2):1–157, 2014. 1, 2, 4

[32] F. Woolfe, E. Liberty, V. Rokhlin, and M. Tygert. A fast randomized algorithm for the approximation of matrices. *Applied and Computational Harmonic Analysis*, 25(3):335–366, 2008. 2