2019

An online algorithm for separating sparse and low-dimensional signal sequences from their sum, and its applications in video processing

Han Guo
Iowa State University

An online algorithm for separating sparse and low-dimensional signal sequences from their sum, and its applications in video processing

by

Han Guo

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Electrical Engineering (Communications and Signal Processing)

Program of Study Committee:
Namrata Vaswani, Major Professor
Chinmay Hegde
Neil Zhenqiang Gong
Jennifer Newman
Songting Luo

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this dissertation. The Graduate College will ensure this dissertation is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2019

DEDICATION

I would like to dedicate this dissertation to my mother Xiaoming Wang and father Bingshan Guo. Without their support I would not have been able to complete this work.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to take this opportunity to express my thanks to those who helped my research directions and the writing of this dissertation. First and foremost, Dr. Namrata Vaswani for her guidance and patience throughout my Ph.D. career. It was my great honor to join her research lab after one year working as a Teaching Assistant in the ECpE department. Her broad knowledge of signal processing and machine learning, strong mathematical skills and sharp research insights greatly influenced my research.

I would like to thank my committee members Dr. Chinmay Hedge, Dr. Neil Zhenqiang Gong, Dr. Jennifer Newman and Dr. Songting Luo. I was inspired by their courses, seminars, presentations and all the office-visits for research discussions. The academic knowledge I learnt from them and the way they conducted research has influenced my Ph.D. study and will make greater impacts in my work after school.

Special thanks to Dr. Vishy Swaminathan and Dr. Saayan Mitra for their guidance during my internships at Adobe Research in 2018 and 2019. Through the internships I got the chance to work on deep-learning related projects and the success on the projects eventually brought me a full-time position at Adobe.

Also, I would like to thank all my wonderful friends at Ames, Iowa. I was lucky enough to meet my piano teacher Mrs. Larisa Kanevski and improved my piano skills over the past five years under her guidance. I cherish my friendships with Li Tang, Jinchun Zhan, Renliang Gu, Yan Ren, Wenyu Wang, Zhengyu Chen, Praneeth Narayanamurthy and all team-members in the research lab.

Finally, I would sincerely express my gratitude to my parents Xiaoming Wang and Bingshan Guo. It was their unconditional love and support that helped me get through all the difficulties in my Ph.D. study and complete this work.

ABSTRACT

In signal processing, "low-rank + sparse" is an important assumption when separating two signals from their sum. Many applications, e.g., video foreground/background separation are well-formulated by this assumption. In this work, with the "low-rank + sparse" assumption, we design and evaluate an online algorithm, called practical recursive projected compressive sensing (prac-ReProCS) for recovering a time sequence of sparse vectors S_t and a time sequence of dense vectors L_t from their sum, $M_t := S_t + L_t$, when the L_t's lie in a slowly changing low-dimensional subspace of the full space.

In the first part of this work (Chapter 1-5), we study and discuss the prac-ReProCS algorithm, the practical version of the original ReProCS algorithm. We apply prac-ReProCS to a key application – video layering, where the goal is to separate a video sequence into a slowly changing background sequence and a sparse foreground sequence that consists of one or more moving regions/objects on-the-fly. Via experiments we show that prac-ReProCS has significantly better performance compared with other state-of-the-art robust-pca methods when applied to video foreground-background separation.

In the second part of this work (Chapter 6), we study the problem of video denoising. We apply prac-ReProCS to video denoising as a preprocessing step. We develop a novel approach to video denoising that is based on the idea that many noisy or corrupted videos can be split into three parts – the "low-rank laye", the "sparse layer" and a small residual which is small and bounded. We show using extensive experiments, layering-then-denoising is effective, especially for long videos with small-sized images that those corrupted by general large variance noise or by large sparse noise, e.g., salt-and-pepper noise.

In the last part of this work (Chapter 7), we discuss an independent problem called logo detection and propose a future research direction where prac-ReProCS can be combined with deep learning solutions.

CHAPTER 1. OVERVIEW

1.1 Introduction

In signal processing, there exists a typical task to recover a time sequence of sparse vectors S_t and a time sequence of dense vectors L_t from their sum, $M_t := S_t + L_t$, when the L_t's lie in a low-rank subspace of $\mathbb{R}^n$.

original foreground background

Figure 1.1: Example of foreground/background separation. First row: original sequence. Second row: foreground sequence. Third row: background image.

A key example of this task is video layering where the goal is to separate a slowly changing background from moving foreground objects/regions [94, 10] as shown in Figure.1.1. By convention, we denote an image by a 1D vector S_t by stacking its columns in the pixel matrix (Figure.1.2). With such representation, a matrix can denote a video with the columns being the image frames (Figure.1.3). In videos that contain foreground/background content, the foreground layer usually consists of one or more moving objects/persons/regions that move in a correlated fashion, i.e. it is a sparse image sequence that often changes in a correlated fashion over time. In most static camera videos, the background images do not change much over time and hence the background

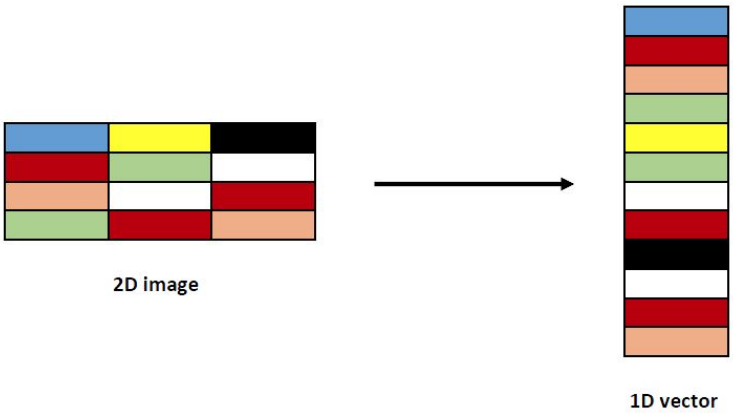


Figure 1.2: Example: stack the columns to form a matrix

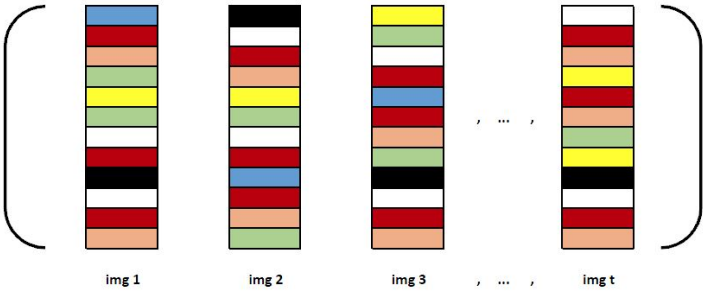


Figure 1.3: Example: representation of a video

image sequence is well modeled as lying in a fixed or slowly-changing low-dimensional subspace of $\mathbb{R}^n$ [101, 10]. An extreme example is when the background is completely static which results in a rank-1 background matrix $[L_1, L_2, \cdots, L_t]$ since all L_t's are identical. Moreover the changes are typically global, e.g. due to lighting variations, and hence modeling it as a dense image sequence is valid too [10].

The magnitude of the entries of L_t could be larger, roughly equal or smaller than that of the nonzero entries of S_t.

The above video layering problem can be interpreted as one of online/recursive sparse recovery from potentially large but structured noise. In this case, S_t is the quantity of interest and L_t is the potentially large but structured low-dimensional noise. The related applications are automatic video surveillance, tracking moving objects, or video conferencing. Alternatively it can be posed

as a recursive/online robust principal components analysis (PCA) problem. In this case L_t, or in fact, the subspace in which it lies, is the quantity of interest while S_t is the outlier, and a typical application is video background editing.

The task of recovering signls S_t and L_t from $M_t := S_t + L_t$ can be extended to the undersampled case, $M_t := AS_t + BL_t$. Applications where this problem occurs include solving the video layering problem from compressive video measurements, e.g., those acquired using a single-pixel camera; online detection of brain activation patterns from undersampled functional MRI (fMRI) sequences (the "active" part of the brain forms the sparse image, while the rest of the brain which does not change much over time forms the low-dimensional part). They can be formulated as $M_t := AS_t + BL_t$ with $B = A$.

Separating two signals from their sum, or signal de-mixing, not only is an important task of recovering the two source signals, but also an effective pre-processing step for many other tasks. In computer vision, many algoritms including object detection and object tracking, have a more robust performance if the given image/video is firstly layered and the algorithm is performed on the foreground mask [93]. Another application where we found layering-first is effective is video denoising. Specifically, if an image frame in a video is corrupted by some noise W_t, i.e., $M_t := S_t + L_t + W_t$ where S_t and L_t are sparse and low-dimensional respectively as described before, the process of layering M_t into two layers actually helps removing the W_t. We dissuss this later in the chapter on video denoising.

1.2 Notation

For a set $T \subseteq \{1, 2, \cdots n\}$, we use $|T|$ to denote its cardinality; and we use T^c to denote its complement, i.e. $T^c := \{i \in \{1, 2, \ldots n\} : i \notin T\}$. The symbols $\cup, \cap, \setminus$ denote set union set intersection and set difference respectively (recall $T_1 \setminus T_2 := T_1 \cap T_2^c$). For a vector v, v_i denotes the ith entry of v and v_T denotes a vector consisting of the entries of v indexed by T. We use $\|v\|_p$ to denote the ℓ_p norm of v. The support of v, $\text{supp}(v)$, is the set of indices at which v is nonzero, $\text{supp}(v) := \{i : v_i \neq 0\}$. We say that v is s-sparse if $|\text{supp}(v)| \leq s$.

For a matrix B, B' denotes its transpose, and $B^\dagger$ denotes its pseudo-inverse. For a matrix with linearly independent columns, $B^\dagger = (B'B)^{-1}B'$. The notation [.] denotes an empty matrix. We use I to denote an identity matrix. For an $m \times n$ matrix B and an index set $T \subseteq \{1, 2, \dots n\}$, B_T is the sub-matrix of B containing columns with indices in the set T. Notice that $B_T = BI_T$. We use $B \setminus B_T$ to denote B_{T^c}. Given another matrix B_2 of size $m \times n_2$, $[B\ B_2]$ constructs a new matrix by concatenating matrices B and B_2 in horizontal direction. Thus, $[(B \setminus B_T)\ B_2] = [B_{T^c}\ B_2]$. We use the notation $B \stackrel{SVD}{=} U\Sigma V'$ to denote the singular value decomposition (SVD) of B with the diagonal entries of Σ being arranged in non-decreasing order.

The interval notation $[t_1, t_2] := \{t_1, t_1 + 1, \cdots, t_2\}$ and similarly the matrix $[L_{t_1}, \dots L_{t_2}] := [L_{t_1}, L_{t_1+1}, \cdots, L_{t_2}]$

Definition 1.2.1 *The* s-restricted isometry constant (RIC) *[8],* δ_s*, for an* $n \times m$ *matrix* Ψ *is the smallest real number satisfying* $(1 - \delta_s)\|x\|_2^2 \leq \|\Psi_T x\|_2^2 \leq (1 + \delta_s)\|x\|_2^2$ *for all sets* T *with* $|T| \leq s$ *and all real vectors* x *of length* $|T|$*.*

Definition 1.2.2 *For a matrix* M*,*

- *range*(M) *denotes the subspace spanned by the columns of* M*.*
- M *is a* basis matrix *if* $M'M = I$*.*
- *The* notation Q = basis(range(M)), or Q = basis(M) for short, *means that* Q *is a* basis matrix *for range*(M) *i.e.* Q *satisfies* $Q'Q = I$ *and range*(Q) = *range*(M)*.*

Definition 1.2.3

- *The* b% left singular values' set *of a matrix* M *is the smallest set of indices of its singular values that contains at least* b% *of the total singular values' energy. In other words, if* $M \stackrel{SVD}{=} U\Sigma V'$*, it is the smallest set* T *such that* $\sum_{i \in T} (\Sigma)_{i,i}^2 \geq \frac{b}{100} \sum_{i=1}^{n} (\Sigma)_{i,i}^2$*.*
- *The corresponding matrix of left singular vectors,* U_T*, is referred to as the* b% left singular vectors' matrix.

- *The notation* $[Q, \Sigma] =$ approx-basis$(M, b\%)$ *means that* Q *is the* $b\%$ *left singular vectors' matrix for* M *and* Σ *is the diagonal matrix with diagonal entries equal to the* $b\%$ *left singular values' set.*

- *The notation* $Q =$ approx-basis(M, r) *means that* Q *contains the left singular vectors of* M *corresponding to its* r *largest singular values. This also sometimes referred to as:* Q contains the r top singular vectors of M.

1.3 Dissertation outline

The remainder of this work is organized as follows:

Chapter 2 presents background knowledge for the studied problem, including a brief introduction of related work and mathematical preliminaries for sparse recovery and compressive sensing.

Chapter 3 introduces the definition of our problem and the three basic assumptions we proposed, which basically require the subspace is slowy-changing and dense, and the support size is small and at each time there should be some change and the change is small.

Chapter 4 formally discusses the main algorithm, prac-ReProCS we developed for separating two signals from their sum. We firstly introduce the basic algorithm which iteratively performs sparse recovery and orthogonal projection. Next we discuss methods to utilize the slow support change and improve support estimation. For subspace update, we introduce two schemes which are simple recursive-PCA and projection-PCA. We conclude the chapter by analyzing the compressive case.

Chapter 5 demonstrates the simulation results and draws conclusions for the prac-ReProCS algorithm. Specically, we firstly describe our method to verifying the model assumptions and the results show that they are valid for real applications such as video foreground/background separation. Next we introduce the experiments for performance comparison. The experiments were with simulated data, partly simulated data and real video data, and all demonstrated that prac-ReProCS has significantly better performance than other compared algorithms.

Chapter 6 starts a new discussion on video denoising, which can be seen as an extended application of prac-ReProCS. We focus on explaining why the idea of laying-first can help in a traditional video denoising task and introduce our ReProCS-based Layering Denoising (ReLD) approach. The extensive experiments show that ReLD is most powerful for videos containing small-sized images and for videos with very large noise.

Chapter 7 concludes this work by the discusson of potential applications of prac-ReProCS and an independent project on logo detection which was done at Adobe Research as an intern project. We demonstrate the experimental results we have obtained and propose a promising future work direction, which tries to combine prac-ReProCS with deep learning approaches to make the current model become fully automatic.

CHAPTER 2. RELATED WORK AND MATHEMATICAL PRELIMINARIES

2.1 Brief review of related work

Most high dimensional data often approximately lie in a lower dimensional subspace. Principal components' analysis (PCA) is a widely used dimension reduction technique that finds a small number of orthogonal basis vectors (principal components), along which most of the variability of the dataset lies. For a given dimension, r, PCA finds the r-dimensional subspace that minimizes the mean squared error between data vectors and their projections into this subspace [39]. It is well known that PCA is very sensitive to outliers. Computing the PCs in the presence of outliers is called robust PCA. Solving the robust PCA problem recursively as more data comes in is referred to as online or recursive robust PCA. "Outlier" is a loosely defined term that usually refers to any corruption that is not small compared to the true signal (or data vector) and that occurs only occasionally. As suggested in [102], an outlier can be nicely modeled as a sparse vector.

In the last few decades, there has been a large amount of work on robust PCA, e.g. [94, 82, 113, 105, 62], and recursive robust PCA e.g. [3, 87, 51]. In most of these works, either the locations of the missing/corruped data points are assumed known [3] (not a practical assumption); or they first detect the corrupted data points and then replace their values using nearby values [87]; or weight each data point in proportion to its reliability (thus soft-detecting and down-weighting the likely outliers) [94, 51]; or just remove the entire outlier vector [105, 62]. Detecting or soft-detecting outliers (S_t) as in [87, 94, 51] is easy when the outlier magnitude is large, but not when it is of the same order or smaller than that of the L_t's.

PCP-based approaches:

In a series of recent works [10, 13], a new and elegant solution to robust PCA called Principal Components' Pursuit (PCP) has been proposed, that does not require a two step outlier location

detection/correction process and also does not throw out the entire vector. It redefines batch robust PCA as a problem of separating a low rank matrix, $\mathcal{L}_t := [L_1, \ldots, L_t]$, from a sparse matrix, $\mathcal{S}_t := [S_1, \ldots, S_t]$, using the measurement matrix, $\mathcal{M}_t := [M_1, \ldots, M_t] = \mathcal{L}_t + \mathcal{S}_t$. Other recent works that also study batch algorithms for recovering a sparse $\mathcal{S}_t$ and a low-rank $\mathcal{L}_t$ from $\mathcal{M}_t := \mathcal{L}_t + \mathcal{S}_t$ or from undersampled measurements include [63, 12, 38, 97, 79, 36, 60, 100, 28, 92]. It was shown in [10] that by solving PCP:

$$\min_{\mathcal{L},\mathcal{S}} \|\mathcal{L}\|_* + \lambda \|\mathcal{S}\|_1 \text{ subject to } \mathcal{L} + \mathcal{S} = \mathcal{M}_t \tag{2.1}$$

one can recover $\mathcal{L}_t$ and $\mathcal{S}_t$ exactly, provided that (a) $\mathcal{L}_t$ is "dense"; (b) any element of the matrix $\mathcal{S}_t$ is nonzero w.p. ϱ, and zero w.p. $1 - \varrho$, independent of all others (in particular, this means that the support sets of the different S_t's are independent over time); and (c) the rank of $\mathcal{L}_t$ and the support size of $\mathcal{S}_t$ are small enough. Here $\|A\|_*$ is the nuclear norm of a matrix A (sum of singular values of A) while $\|A\|_1$ is the ℓ_1 norm of A seen as a long vector.

Specifically, (a) and (b) are required to avoid the identifiability issue – $\mathcal{L}_t$ should not be sparse and $\mathcal{S}_t$ not low-rank. Write the singular value decomposition of $\mathcal{L}_t$ (of size $n_1 \times n_2$) as $\mathcal{L}_t = \mathcal{U}\Sigma\mathcal{V}^* = \sum_{i=1}^{r} \sigma_i U_i V_i^*$, where r is the rank of the matrix, $\sigma_i, \ldots, \sigma_r$ are the positive singular values, and $\mathcal{U} = [U_1, \ldots, U_r]$, $\mathrm{V} = [V_1, \ldots, V_r]$ are the matrices of left and right-singular vectors. It is required that the singular vectors are reasonably spreadout by restricting

$$\max_i \|\mathcal{U}^* e_i\|^2 \leq \frac{\mu r}{n_1}, \max_i \|\mathcal{V}^* e_i\|^2 \leq \frac{\mu r}{n_2}, \tag{2.2}$$

and

$$\|\mathcal{U}\mathcal{V}^*\|_\infty \leq \sqrt{\frac{\mu r}{n_1 n_2}} \tag{2.3}$$

for a given μ.

PCP tries to minimize the rank of $\mathcal{L}$ and sparsity of $\mathcal{S}$ with a convex approach and it is solvable in polynomial time. The art here is that the nuclear norm serves as a convex surrogate for the rank of a matrix and the ℓ_1 norm of a vectorized matrix serves as a convex surrogate for the support size.

Non-convex approaches:

It has been analyzed that the number of iteratons needed for a convex program solver to get to within an ϵ ball of the true solution of the convex program is $O(\frac{1}{\epsilon})$, which makes the typical complexity for a PCP solver $O(\frac{nd^2}{\epsilon})$ (suppose the matrix is of size $n \times d$) [66]. To address this issue, more recent non-convex solutions were proposed which are provably mush faster. Two representing solutions are:

- Alternating-Projection (AltProj) [66]:

AltProj tries to find a matrix $\mathcal{L}$ that lies in the intersection of two sets: $L_{\text{set}} = \{\text{set of rank-}r \text{ matrics}\}$ and $S_{\text{set}} = \{\mathcal{M}_t - \mathcal{S}, \text{where } \mathcal{S} \text{ is a sparse matrix}\}$. The algorithms alternately projects onto these two non-convex sets, while appropriately relaxing the rank and the sparsity levels. It needs time of order $O(ndr^2\text{log}(1/\epsilon))$ to achieve the error $\|\hat{\mathcal{L}} - \mathcal{L}\|_F \leq \epsilon$ and $\|\hat{\mathcal{S}} - \mathcal{S}\|_\infty \leq \epsilon$ provided that (a) $\mathcal{U}, \mathcal{V}$ are μ-incoherent and (b) each row and column of $\mathcal{S}$ have at most α fraction of non-zero entries such that $\alpha \leq \frac{1}{c\mu^2 r}$. Here $\mathcal{U}, \mathcal{V}$ are the matrices of left and right-singular vectors of $\mathcal{L}$.

- Projected Gradient Descent (RPCA-GD) [108]:

RPCA-GD needs time of order $O(ndr^{\log}(1/\epsilon))$ to achieve the error $\|\hat{\mathcal{L}} - \mathcal{L}\|_F \leq \epsilon\sigma_{\max}(\mathcal{L})$ provided provided that (a) $\mathcal{U}, \mathcal{V}$ are μ-incoherent and (b) each row and column of S have at most α fraction of non-zero entries such that $\alpha \leq \frac{c}{\mu r}$. Here $\mathcal{U}, \mathcal{V}$ are the matrices of left and right-singular vectors of $\mathcal{L}$. It also provided the solution for the robust PCA problem with partial observations.

Online RPCA. Notice that most robust-pca based applications e.g., video foreground/background separation require an online solution. A batch solution would need a long delay; and would also be much slower and more memory-intensive than a recursive solution. Moreover, the assumption that the foreground support is independent over time (required by PCP) is not usually valid. To address these issues, in the conference versions of the work [71, 72], a recursive solution called Recursive Projected Compressive Sensing (ReProCS) was introduced. In recent work [74, 73, 1], performance guarantees for ReProCS have been obtained. Under mild assumptions (denseness, slow enough subspace change of ℓ_t and "some" support change at least every h frames of s_t), it has been shown that, with high probability (w.h.p.), ReProCS can exactly recover the support set of S_t at all times;

and the reconstruction errors of both S_t and L_t are upper bounded by a time invariant and small value. The work of [74, 73] contains a partial result while [1] is a complete correctness result.

Other very recent work on recursive / online robust PCA includes [33, 25, 24, 61, 59]. Some other related work includes work that uses structured sparsity models, e.g. [43]. For our problem, if it is known that the sparse vector consists of one or a few connected regions, these ideas could be incorporated into our algorithm as well. On the other hand, the advantage of the current approach that only uses sparsity is that it works both for the case of a few connected regions as well as for the case of multiple small sized moving objects, e.g. see the airport video results at http://www.ece.iastate.edu/~chenlu/ReProCS/Video_ReProCS.htm.

2.2 Mathematical preliminaries

We provided the basic compressive sensing result here as needed in our main algorithm.

2.2.1 Compressive sensing results

Compressive sensing is a signal processing technique for efficiently acquiring and reconstructing a signal, by finding solutions to undertermined linear systems. Through optimization, the sparsity of a signal can be explited to recover it from only a few samples. It is required that the signal must be sparse in some domain.

Theorem 2.2.1 *[7] For a vector x, use x_s to denote a vector with all but the s-largest entries set to zero. Suppose we observe*

$$y := \Psi x. \tag{2.4}$$

Let $\hat{x}$ be the solution to following problem

$$\min_x \|x\|_1 \text{ subject to } y = \Psi x \tag{2.5}$$

Assume that $\delta_{2s} < \sqrt{2} - 1$, then $\hat{x}$ obeys

$$\|\hat{x} - x\|_1 \leq C_0 \|x - x_s\|_1 \tag{2.6}$$

and

$$\|\hat{x} - x\|_2 \leq C_0 s^{-\frac{1}{2}} \|x - x_s\|_1 \tag{2.7}$$

for some constant C_0. In particular, if x is s-sparse, the recovery is exact.

Theorem 2.2.2 *[7]*

Suppose we observe

$$y := \Psi x + z \tag{2.8}$$

with z being the noise. Let $\hat{x}$ be the solution to following problem

$$\min_x \|x\|_1 \text{ subject to } \|y - \Psi x\|_2 \leq \xi. \tag{2.9}$$

Assume that x is s-sparse, $\|z\|_2 \leq \xi$ and $\delta_{2s}(\Psi) < b(\sqrt{2} - 1)$ with $0 \leq b < 1$. Then the solution of (2.9) obeys

$$\|\hat{x} - x\|_2 \leq C_1 \xi \tag{2.10}$$

for some constant C_1

2.2.2 Modified compressive sensing

Modified-CS [96] was a solution to the problem of sparse reconstruction with partial knowledge of the support. Let the "known" support be $\mathcal{T}$. Modified-CS tries to find a signal that is sparses outside of the set $\mathcal{T}$ among all signals satisfying the data constraint.

Specifically, modified-CS solves

$$\min_x \|x_{\mathcal{T}^c}\|_1 \text{ s.t. } \|y - \Psi x\|_2 \leq \xi. \tag{2.11}$$

For recursively reconstructing a time sequence of sparse signals, the support estimate from the previous time, $\hat{\mathcal{N}}_{t-1}$ can be incorparated as a prior for the current support. By doing this, modified-CS is often formulated by solving

$$\min_x \|x_{\mathcal{T}^c}\|_1 \text{ s.t. } \|y - \Psi x\|_2 \leq \xi, \quad \hat{\mathcal{T}} = \hat{\mathcal{N}}_{t-1}. \tag{2.12}$$

In fact, modified-CS belongs to a broader idea called weighted-ℓ_1 minimization where the entries of x use two different weights, i.e.,

$$\min_x \lambda \|x_{\mathcal{T}}\|_1 + \|x_{\mathcal{T}^c}\|_1 \text{ s.t. } \|y_t - \Phi_t x\|_2 \leq \xi \tag{2.13}$$

We can see that modified-CS is a special case by setting the weight on $\mathcal{T}$ to 0.

CHAPTER 3. PROBLEM DEFINITION AND ASSUMPTIONS

We suppose that, the measurement vector at time t, M_t, is an n dimensional vector which can be decomposed as

$$M_t := S_t + L_t. \tag{3.1}$$

Let T_t denote the support set of S_t, i.e.,

$$T_t := \text{supp}(S_t) = \{i : (S_t)_i \neq 0\}.$$

We assume that S_t and L_t satisfy the assumptions given below in the next three subsections. Suppose that an initial training sequence which does not contain the sparse components is available, i.e. we are given $\mathcal{M}_{\text{train}} = [M_t; 1 \leq t \leq t_{\text{train}}]$ with $M_t = L_t$. This is used to get an initial estimate of the subspace in which the L_t's lie. If an initial sequence without S_t's is not available, one can use a batch robust PCA algorithm to get the initial subspace estimate as long as the initial sequence satisfies its required assumptions. At each $t > t_{\text{train}}$, the goal is to recursively estimate S_t and L_t and the subspace in which L_t lies. By "recursively" we mean: use $\hat{S}_{t-1}, \hat{L}_{t-1}$ and the previous subspace estimate to estimate S_t and L_t.

The magnitude of the entries of L_t may be small, of the same order, or large compared to that of the nonzero entries of S_t. In applications where S_t is the signal of interest, the case when $\|L_t\|_2$ is of the same order or larger than $\|S_t\|_2$ is the difficult case.

A key application where the above problem occurs is in separating a video sequence into background and foreground layers. Let Im_t denote the image at time t, F_t denote the foreground image at t and B_t the background image at t, all arranged as 1-D vectors. Then, the image sequence satisfies

$$(\text{Im}_t)_i = \begin{cases} (F_t)_i & \text{if } i \in \text{supp}(F_t) \\ (B_t)_i & \text{if } i \notin \text{supp}(F_t) \end{cases} \tag{3.2}$$

In fMRI, F_t is the sparse active region image while B_t is the background brain image. In both cases, it is fair to assume that an initial background-only training sequence is available. For video this means there are no moving objects/regions in the foreground. For fMRI, this means some frames are captured without providing any stimulus to the subject.

Let μ denote the empirical mean of the training background images. If we let $L_t := B_t - \mu$, $M_t := \mathrm{Im}_t - \mu$, $T_t := \mathrm{supp}(F_t)$, and

$$(S_t)_{T_t} := (F_t - B_t)_{T_t},\ (S_t)_{T_t^c} := 0,$$

then, clearly, $M_t = S_t + L_t$. Once we get the estimates $\hat{L}_t$, $\hat{S}_t$, we can also recover the foreground and background as

$$\hat{B}_t = \hat{L}_t + \mu,\ \hat{T}_t = \mathrm{supp}(\hat{S}_t),\ (\hat{F}_t)_{\hat{T}_t} = (\mathrm{Im}_t)_{\hat{T}_t},\ (\hat{F}_t)_{\hat{T}_t^c} = 0.$$

3.1 Slowly changing low-dimensional subspace change

We assume that for τ large enough, any τ length subsequence of the L_t's lies in a subspace of $\mathbf{R}^n$ of dimension less than $\min(\tau, n)$, and usually much less than $\min(\tau, n)$. In other words, for τ large enough, $\max_t \mathrm{rank}([L_{t-\tau+1}, \dots L_t]) \ll \min(\tau, n)$. Also, this subspace is either fixed or changes slowly over time.

One way to model this is as follows [74]. Let $L_t = P_t a_t$ where P_t is an $n \times r_t$ *basis matrix* with $r_t \ll n$ that is piecewise constant with time, i.e. $P_t = P_{(j)}$ for all $t \in [t_j, t_{j+1})$ and $P_{(j)}$ changes as

$$P_{(j)} = [(P_{(j-1)} R_j \setminus P_{(j),\mathrm{old}}), P_{(j),\mathrm{new}}]$$

where $P_{(j),\mathrm{new}}$ and $P_{(j),\mathrm{old}}$ are basis matrices of size $n \times c_{j,\mathrm{new}}$ and $n \times c_{j,\mathrm{old}}$ respectively with $P'_{(j),\mathrm{new}} P_{(j-1)} = 0$ and R_j is a rotation matrix. Moreover, (a) $0 \le \sum_{i=1}^{j} (c_{i,\mathrm{new}} - c_{i,\mathrm{old}}) \le c_{\mathrm{dif}}$; (b) $0 \le c_{j,\mathrm{new}} \le c_{\max} < r_0$; (c) $(t_{j+1} - t_j) \gg r_0 + c_{\mathrm{dif}}$; and (d) there are a total of J change times with $J \ll (n - r_0 - c_{\mathrm{dif}})/c_{\max}$.

Clearly, (a) implies that $r_t \le r_0 + c_{\mathrm{dif}} := r_{\max}$ and (d) implies that $r_{\max} + J c_{\max} \ll n$. This, along with (b) and (c), helps to ensure that for any $\tau > r_{\max} + c_{\max}$, $r^{t,\tau} := \max_t \mathrm{rank}([L_{t-\tau+1}, \dots L_t]) < \min(\tau, n)$, and for $\tau \gg r_{\max} + c_{\max}$, $r^{t,\tau} \ll \min(\tau, n)$

Notice first that (c) implies that $(t_{j+1}-t_j) \gg r_{\max}$. Also, (b) implies that $\text{rank}([L_{t_j}, \dots L_{t_{j+k}-1}]) \leq r_{\max} + (k-1)c_{\max}$. First consider the case when both $t-\tau+1$ and t lie in $[t_j, t_{j+1}-1]$. In this case, $r^{t,\tau} \leq r_{\max}$ for any τ. Thus for any $t_{j+1}-t_j > \tau \gg r_{\max}$, $r^{t,\tau} \ll \min(\tau, n)$. Next consider the case when $t-\tau+1 \in [t_j, t_{j+1}-1]$ and $t \in [t_{j+1}, t_{j+2}-1]$. In this case, $r^{t,\tau} \leq r_{\max} + c_{\max}$. Thus, for any $t_{j+2}-t_j > \tau \gg r_{\max}+c_{\max}$, $r^{t,\tau} \ll \min(\tau, n)$. Finally consider the case when $t-\tau+1 \in [t_j, t_{j+1}-1]$ and $t \in [t_{j+k+1}, t_{j+k+2}-1]$ for a $0 < k < J-1$. In this case, τ can be rewritten as $\tau = (t_{j+k+1}-t_{j+1}) + \tau_1 + \tau_2$ with $\tau_1 := t_{j+1} - (t-\tau+1)$ and $\tau_2 := t - (t_{j+k+1}-1)$. Clearly, $r^{t,\tau} \leq (r_{\max}+(k-1)c_{\max}) + \min(\tau_1, c_{\max}) + \min(\tau_2, c_{\max}) < kr_{\max} + \min(\tau_1, c_{\max}) + \min(\tau_2, c_{\max}) \ll (t_{j+k+2} - t_{j+1}) + \min(\tau_1, c_{\max}) + \min(\tau_2, c_{\max}) \leq (t_{j+k+2} - t_{j+1}) + \tau_1 + \tau_2 = \tau$. Moreover, $r^{t,\tau} \leq r_{\max} + (k+1)c_{\max} \leq r_{\max} + Jc_{\max} \ll n$. Thus, in this case again for any τ, $r^{t,\tau} \ll \min(\tau, n)$.

By slow subspace change, we mean that: for $t \in [t_j, t_{j+1})$, $\|(I - P_{(j-1)}P'_{(j-1)})L_t\|_2$ is initially small and increases gradually. In particular, we assume that, for $t \in [t_j, t_j + \alpha)$,

$$\|(I - P_{(j-1)}P'_{(j-1)})L_t\|_2 \leq \gamma_{\text{new}} \ll \min(\|L_t\|_2, \|S_t\|_2)$$

and increases gradually after $t_j + \alpha$. One model for "increases gradually" is as given in [74, Sec III-B]. Nothing in this work requires the specific model and hence we do not repeat it here.

The above piecewise constant subspace change model is a simplified model for what typically happens in practice. In most cases, P_t changes a little at each t in such a way that the low-dimensional assumption approximately holds. If we try to model this, it would result in a non-stationary model that is difficult to precisely define or to verify (it would require multiple video sequences of the same type to verify) [1].

Since background images typically change only a little over time (except in case of a camera viewpoint change or a scene change), it is valid to model the mean-subtracted background image sequence as lying in a slowly changing low-dimensional subspace. We verify this assumption in Sec 5.1.

[1] With letting a_t be a zero mean random variable with a covariance matrix that is constant for sub-intervals within $[t_j, t_{j+1})$, the above model is a piecewise wide sense stationary approximation to the nonstationary model.

3.2 Denseness assumption

To state the denseness assumption, we first need to define the denseness coefficient. This is a simplification of the one introduced in [74].

Definition 3.2.1 (denseness coefficient) *For a matrix or a vector B, define*

$$\kappa_s(B) = \kappa_s(range(B)) := \max_{|T| \leq s} \|I_T{}' basis(B)\|_2 \tag{3.3}$$

where $\|.\|_2$ is the vector or matrix 2-norm. Recall that basis(B) is short for basis(range(B)). Similarly $\kappa_s(B)$ is short for $\kappa_s(range(B))$. Notice that $\kappa_s(B)$ is a property of the subspace range(B). Note also that $\kappa_s(B)$ is a non-decreasing function of s and of rank(B).

We assume that the subspace spanned by the L_t's is dense, i.e.

$$\kappa_{2s}(P_{(j)}) = \kappa_{2s}([L_{t_j}, \ldots L_{t_{j+1}-1}]) \leq \kappa_*$$

for a κ_* significantly smaller than one. Moreover, a similar assumption holds for $P_{(j),\text{new}}$ with a tighter bound: $\kappa_{2s}(P_{(j),\text{new}}) \leq \kappa_{\text{new}} < \kappa_*$. This assumption is similar to one of the denseness assumptions used in [11, 10]. In [10], a bound is assumed on $\kappa_1(U)$ and $\kappa_1(V)$ where U and V are the matrices containing the left and right singular vectors of the entire matrix, $[L_1, L_2 \ldots L_t]$; and a tighter bound is assumed on $\max_{i,j} |(UV')_{i,j}|$. In our notation, $U = [P_{(0)}, P_{(1),\text{new}}, \ldots P_{(J),\text{new}}]$.

The following lemma, proved in [74], relates the RIC of $I - PP'$, when P is a *basis matrix*, to the denseness coefficient for range(P). Notice that $I - PP'$ is an $n \times n$ matrix that has rank $(n - \text{rank}(P))$ and so it cannot be inverted.

Lemma 3.2.2 *For a* basis matrix*, P,*

$$\delta_s(I - PP') = \kappa_s(P)^2$$

Thus, the denseness assumption implies that the RIC of the matrix $(I - P_{(j)}P'_{(j)})$ is small. Using any of the RIC based sparse recovery results, e.g. [7], this ensures that for $t \in [t_j, t_{j+1})$, s-sparse vectors S_t are recoverable from $(I - P_{(j)}P'_{(j)})M_t = (I - P_{(j)}P'_{(j)})S_t$ by ℓ_1 minimization.

Very often, the background images primarily change due to lighting changes (in case of indoor sequences) or due to moving waters or moving leaves (in case of many outdoor sequences) [10, 74]. All of these result in global changes and hence it is valid to assume that the subspace spanned by the background image sequences is dense.

3.3 Small support size, some support change, small support change assumption on S_t

Let the sets of support additions and removals be

$$\Delta_t := T_t \setminus T_{t-1},\ \Delta_{e,t} := T_{t-1} \setminus T_t.$$

(1) We assume that

$$|T_t| + \min(|T_t|, |\Delta_t| + |\Delta_{e,t}|) \leq s + s_\Delta \text{ where } s_\Delta \ll s$$

In particular, this implies that we either need $|T_t| \leq s$ and $|\Delta_t| + |\Delta_{e,t}| \leq s_\Delta$ (S_t is sparse with support size at most s, and its support changes slowly) or, in cases when the change $|\Delta_t| + |\Delta_{e,t}|$ is large, we need $|T_t| \leq 0.5(s + s_\Delta)$ (need a tighter bound on the support size).

(2) We also assume that there is *some* support change every few frames, i.e. at least once every h frames, $|\Delta_t| > s_{\Delta,\min}$. Practically, this is needed to ensure that at least some of the background behind the foreground is visible so that the changes to the background subspace can be estimated.

In video applications, foreground images often consist of one or more moving objects/people/regions and hence are sparse. Also, typically the objects are not static, i.e. there is some support change at least every few frames. On the other hand, since the objects usually do not move very fast, slow support change is also valid most of the time. The time when the support change is almost comparable to the support size is usually when the object is entering or leaving the image, but these are the exactly the times when the object's support size is itself small (being smaller than $0.5(s + s_\Delta)$ is a valid).

CHAPTER 4. PRAC-REPROCS: PRACTICAL REPROCS ALGORITHM

We first develop a practical algorithm based on the basic ReProCS idea from earlier work [74]. Then we discuss how the sparse recovery and support estimation steps can be improved. The complete algorithm is summarized in Algorithm 1. Finally we discuss an alternate subspace update procedure in Sec 4.4.

4.1 Basic algorithm

We use $\hat{S}_t, \hat{T}_t, \hat{L}_t$ to denote estimates of S_t, its support, T_t, and L_t respectively; and we use $\hat{P}_t$ to denote the *basis matrix* for the estimated subspace of L_t at time t. Also, let

$$\Phi_t := (I - \hat{P}_{t-1}\hat{P}_{t-1}') \tag{4.1}$$

Given the initial training sequence which does not contain the sparse components, $\mathcal{M}_{\text{train}} = [L_1, L_2, \dots L_{t_{\text{train}}}]$ we compute $\hat{P}_0$ as an approximate basis for $\mathcal{M}_{\text{train}}$, i.e. $\hat{P}_0 = \text{approx-basis}(\mathcal{M}_{\text{train}}, b\%)$. Let $\hat{r} = \text{rank}(\hat{P}_0)$. We need to compute an approximate basis because for real data, the L_t's are only approximately low-dimensional. We use $b\% = 95\%$ or $b\% = 99.99\%$ depending on whether the low-rank part is approximately low-rank or almost exactly low-rank. After this, at each time t, ReProCS involves 4 steps: (a) Perpendicular Projection; (b) Sparse Recovery (recover T_t and S_t); (c) Recover L_t; (d) Subspace Update (update $\hat{P}_t$).

Perpendicular Projection. In the first step, at time t, we project the measurement vector, M_t, into the space orthogonal to range$(\hat{P}_{t-1})$ to get the projected measurement vector,

$$y_t := \Phi_t M_t. \tag{4.2}$$

Sparse Recovery (Recover T_t and S_t). With the above projection, y_t can be rewritten as

$$y_t = \Phi_t S_t + \beta_t \text{ where } \beta_t := \Phi_t L_t \tag{4.3}$$

Because of the slow subspace change assumption, projecting orthogonal to range($\hat{P}_{t-1}$) nullifies most of the contribution of L_t and hence β_t can be interpreted as small "noise". We explain this in detail in APPENDIX .

Thus, the problem of recovering S_t from y_t becomes a traditional noisy sparse recovery/CS problem. Notice that, since the $n \times n$ projection matrix, Φ_t, has rank $n - \text{rank}(\hat{P}_{t-1})$, therefore y_t has only this many "effective" measurements, even though its length is n. To recover S_t from y_t, one can use ℓ_1 minimization [16, 7], or any of the greedy or iterative thresholding algorithms from literature. In this work we use ℓ_1 minimization: we solve

$$\min_x \|x\|_1 \text{ s.t. } \|y_t - \Phi_t x\|_2 \leq \xi \tag{4.4}$$

and denote its solution by $\hat{S}_{t,cs}$. By the denseness assumption, P_{t-1} is dense. Since $\hat{P}_{t-1}$ approximates it, this is true for $\hat{P}_{t-1}$ as well [74, Lemma 6.6]. Thus, by Lemma 3.2.2, the RIC of Φ_t is small enough. Using [7, Theorem 1], this and the fact that β_t is small ensures that S_t can be accurately recovered from y_t. The constraint ξ used in the minimization should equal $\|\beta_t\|_2$ or its upper bound. Since β_t is unknown we set $\xi = \|\hat{\beta}_t\|_2$ where $\hat{\beta}_t := \Phi_t \hat{L}_{t-1}$.

By thresholding on $\hat{S}_{t,cs}$ to get an estimate of its support followed by computing a least squares (LS) estimate of S_t on the estimated support and setting it to zero everywhere else, we can get a more accurate estimate, $\hat{S}_t$, as suggested in [9]. We discuss better support estimation and its parameter setting in Sec 4.3.

Recover L_t. The estimate $\hat{S}_t$ is used to estimate L_t as $\hat{L}_t = M_t - \hat{S}_t$. Thus, if S_t is recovered accurately, so will L_t.

Subspace Update (Update $\hat{P}_t$). Within a short delay after every subspace change time, one needs to update the subspace estimate, $\hat{P}_t$. To do this in a provably reliable fashion, we introduced the projection PCA (p-PCA) algorithm in [74]. The algorithm studied there used knowledge of the subspace change times t_j and of the number of new directions $c_{j,\text{new}}$. Let $\hat{P}_{(j-1)}$ denote the final estimate of a basis for the span of $P_{(j-1)}$. It is assumed that the delay between change times is large enough so that $\hat{P}_{(j-1)}$ is an accurate estimate. At $t = t_j + \alpha - 1$, p-PCA gets the first estimate of the new directions, $\hat{P}_{(j),\text{new},1}$, by projecting the last α $\hat{L}_t$'s perpendicular to $\hat{P}_{(j-1)}$

followed by computing the $c_{j,\text{new}}$ top left singular vectors of the projected data matrix. It then updates the subspace estimate as $\hat{P}_t = [\hat{P}_{(j-1)}, \hat{P}_{(j),\text{new},1}]$. The same procedure is repeated at every $t = t_j + k\alpha - 1$ for $k = 2, 3, \dots K$ and each time we update the subspace as $\hat{P}_t = [\hat{P}_{(j-1)}, \hat{P}_{(j),\text{new},k}]$. Here K is chosen so that the subspace estimation error decays down to a small enough value within K p-PCA steps.

In this paper, we design a practical version of p-PCA which does not need knowledge of t_j or $c_{j,\text{new}}$. This is summarized in Algorithm 1. The key idea is as follows. We let $\hat{\sigma}_{\min}$ be the $\hat{r}^{th}$ largest singular value of the training dataset. This serves as the noise threshold for approximately low rank data. We split projection PCA into two phases: "detect subspace change" and "p-PCA". We are in the detect phase when the previous subspace has been accurately estimated. Denote the basis matrix for this subspace by $\hat{P}_{(j-1)}$. We detect the subspace change as follows. Every α frames, we project the last α $\hat{L}_t$'s perpendicular to $\hat{P}_{(j-1)}$ and compute the SVD of the resulting matrix. If there are any singular values above $\hat{\sigma}_{\min}$, this means that the subspace has changed. At this point, we enter the "p-PCA" phase. In this phase, we repeat the K p-PCA steps described above with the following change: we estimate $c_{j,\text{new}}$ as the number of singular values above $\hat{\sigma}_{\min}$, but clipped at $\lceil \alpha/3 \rceil$ (i.e. if the number is more than $\lceil \alpha/3 \rceil$ then we clip it to $\lceil \alpha/3 \rceil$). We stop either when the stopping criterion given in step 4(b)iv is achieved ($k \geq K_{\min}$ and the projection of $\hat{L}_t$ along $\hat{P}_{\text{new},k}$ is not too different from that along $\hat{P}_{\text{new},k}$) or when $k \geq K_{\max}$.

For the above algorithm, with theoretically motivated choices of algorithm parameters, under the assumptions from chapter 3, it is possible to show that, w.h.p., the support of S_t is exactly recovered, the subspace of L_t's is accurately recovered within a finite delay of the change time. We provide a brief overview of the proof from [74, 1] in APPENDIX that helps explain why the above approach works.

Remark 4.1.1 *The p-PCA algorithm only allows addition of new directions. If the goal is to estimate the span of* $[L_1, \dots L_t]$*, then this is what is needed. If the goal is sparse recovery, then one can get a smaller rank estimate of* $\hat{P}_t$ *by also including a step to delete the span of the removed directions,* $P_{(j),old}$*. This will result in more "effective" measurements available for the sparse re-*

covery step and hence possibly in improved performance. The simplest way to do this is to do one simple PCA step every some frames. In our experiments, this did not help much though. A provably accurate solution is described in [74, Sec VII].

Remark 4.1.2 *The p-PCA algorithm works on small batches of α frames. This can be made fully recursive if we compute the SVD of $(I - \hat{P}_{(j-1)}\hat{P}'_{(j-1)})[\hat{L}_{t-\alpha+1}, \ldots \hat{L}_t]$ using the incremental SVD (inc-SVD) procedure summarized in Algorithm 2 [3] for one frame at a time. As explained in [3] and references therein, we can get the left singular vectors and singular values of any matrix $M = [M_1, M_2, \ldots M_\alpha]$ recursively by starting with $\hat{P} = [.], \hat{\Sigma} = [.]$ and calling [$\hat{P}$, $\hat{\Sigma}$] = inc-SVD($\hat{P}$, $\hat{\Sigma}$, M_i) for every column i or for short batches of columns of size of α/k. Since we use $\alpha = 20$ which is a small value, the use of incremental SVD does not speed up the algorithm in practice and hence we do not report results using it.*

4.2 Exploiting slow support change when valid

[74, 73] used ℓ_1 minimization followed by thresholding and LS for sparse recovery. However if slow support change holds, one can replace simple ℓ_1 minimization by modified-CS [96] which requires fewer measurements for exact/accurate recovery as long as the previous support estimate, $\hat{T}_{t-1}$, is an accurate enough predictor of the current support, T_t. In our application, $\hat{T}_{t-1}$ is likely to contain a significant number of extras and in this case, a better idea is to solve the following weighted ℓ_1 problem [48, 27]

$$\min_x \lambda \|x_T\|_1 + \|x_{T^c}\|_1 \text{ s.t. } \|y_t - \Phi_t x\|_2 \le \xi, \quad T := \hat{T}_{t-1} \tag{4.5}$$

with $\lambda < 1$ (modified-CS solves the above with $\lambda = 0$). Denote its solution by $\hat{S}_{t,cs}$. One way to pick λ is to let it be proportional to the estimate of the percentage of extras in $\hat{T}_{t-1}$. If slow support change does not hold, the previous support estimate is not a good predictor of the current support. In this case, doing the above is a bad idea and one should instead solve simple ℓ_1, i.e. solve (4.5) with $\lambda = 1$. As explained in [27], if the support estimate contains at least 50% correct entries, then weighted ℓ_1 is better than simple ℓ_1. We use the above criteria with true values replaced by

Algorithm 1 Practical ReProCS-pPCA

Input: M_t; **Output:** $\hat{T}_t$, $\hat{S}_t$, $\hat{L}_t$; **Parameters:** $q, b, \alpha, K_{\min}, K_{\max}$. We used $\alpha = 20, K_{\min} = 3, K_{\max} = 10$ in all experiments (α needs to only be large compared to $c_{\max}$); we used $b = 95$ for approximately low-rank data (all real videos and the lake video with simulated foreground) and used $b = 99.99$ for almost exactly low rank data (simulated data); we used $q = 1$ whenever $\|S_t\|_2$ was of the same order or larger than $\|L_t\|_2$ (all real videos and the lake video) and used $q = 0.25$ when it was much smaller (simulated data with small magnitude S_t).
Initialization

- $[\hat{P}_0, \hat{\Sigma}_0] \leftarrow$ approx-basis($\frac{1}{\sqrt{t_{\text{train}}}}[M_1, \dots M_{t_{\text{train}}}], b\%$).
- Set $\hat{r} \leftarrow \text{rank}(\hat{P}_0)$, $\hat{\sigma}_{\min} \leftarrow ((\hat{\Sigma}_0)_{\hat{r},\hat{r}})$, $\hat{t}_0 = t_{\text{train}}$, flag = detect
- Initialize $\hat{P}_{(t_{\text{train}})} \leftarrow \hat{P}_0$ and $\hat{T}_t \leftarrow [.]$.

For $t > t_{\text{train}}$ do

1. Perpendicular Projection: compute $y_t \leftarrow \Phi_t M_t$ with $\Phi_t \leftarrow I - \hat{P}_{t-1}\hat{P}_{t-1}'$
2. Sparse Recovery (Recover S_t and T_t)
   (a) If $\frac{|\hat{T}_{t-2} \cap \hat{T}_{t-1}|}{|\hat{T}_{t-2}|} < 0.5$
      i. Compute $\hat{S}_{t,\text{cs}}$ as the solution of (2(a)ii) with $\xi = \|\Phi_t \hat{L}_{t-1}\|_2$.
      ii. $\hat{T}_t \leftarrow \text{Thresh}(\hat{S}_{t,\text{cs}}, \omega)$ with $\omega = q\sqrt{\|M_t\|^2/n}$. Here $T \leftarrow \text{Thresh}(x, \omega)$ means that $T = \{i : |(x)_i| \geq \omega\}$.
   Else
      i. Compute $\hat{S}_{t,\text{cs}}$ as the solution of (4.5) with $T = \hat{T}_{t-1}$, $\lambda = \frac{|\hat{T}_{t-2} \setminus \hat{T}_{t-1}|}{|\hat{T}_{t-1}|}$, $\xi = \|\Phi_t \hat{L}_{t-1}\|_2$.
      ii. $\hat{T}_{\text{add}} \leftarrow \text{Prune}(\hat{S}_{t,\text{cs}}, 1.4|\hat{T}_{t-1}|)$. Here $T \leftarrow \text{Prune}(x, k)$ returns indices of the k largest magnitude elements of x.
      iii. $\hat{S}_{t,\text{add}} \leftarrow \text{LS}(y_t, \Phi_t, \hat{T}_{\text{add}})$. Here $\hat{x} \leftarrow \text{LS}(y, A, T)$ means that $\hat{x}_T = (A_T{}' A_T)^{-1} A_T{}' y$ and $\hat{x}_{T^c} = 0$.
      iv. $\hat{T}_t \leftarrow \text{Thresh}(\hat{S}_{t,\text{add}}, \omega)$ with $\omega = q\sqrt{\|M_t\|^2/n}$.
   (b) $\hat{S}_t \leftarrow \text{LS}(y_t, \Phi_t, \hat{T}_t)$
3. Estimate L_t: $\hat{L}_t \leftarrow M_t - \hat{S}_t$
4. Update $\hat{P}_t$: projection PCA
   (a) If flag = detect and $\text{mod}(t - \hat{t}_j + 1, \alpha) = 0$, (here $\text{mod}(t, \alpha)$ is the remainder when t is divided by α)
      i. compute the SVD of $\frac{1}{\sqrt{\alpha}}(I - \hat{P}_{(j-1)}\hat{P}_{(j-1)}')[\hat{L}_{t-\alpha+1}, \dots \hat{L}_t]$ and check if any singular values are above $\hat{\sigma}_{\min}$
      ii. if the above number is more than zero then set flag $\leftarrow$ pPCA, increment $j \leftarrow j + 1$, set $\hat{t}_j \leftarrow t - \alpha + 1$, reset $k \leftarrow 1$
   Else $\hat{P}_t \leftarrow \hat{P}_{t-1}$.
   (b) If flag = pPCA and $\text{mod}(t - \hat{t}_j + 1, \alpha) = 0$,
      i. compute the SVD of $\frac{1}{\sqrt{\alpha}}(I - \hat{P}_{(j-1)}\hat{P}_{(j-1)}')[\hat{L}_{t-\alpha+1}, \dots \hat{L}_t]$,
      ii. let $\hat{P}_{j,\text{new},k}$ retain all its left singular vectors with singular values above $\hat{\sigma}_{\min}$ or all $\alpha/3$ top left singular vectors whichever is smaller,
      iii. update $\hat{P}_t \leftarrow [\hat{P}_{(j-1)}\ \hat{P}_{j,\text{new},k}]$, increment $k \leftarrow k + 1$
      iv. If $k \geq K_{\min}$ and $\frac{\|\sum_{t-\alpha+1}^{t}(\hat{P}_{j,\text{new},i-1}\hat{P}_{j,\text{new},i-1}' - \hat{P}_{j,\text{new},i}\hat{P}_{j,\text{new},i}')L_t\|_2}{\|\sum_{t-\alpha+1}^{t}\hat{P}_{j,\text{new},i-1}\hat{P}_{j,\text{new},i-1}'L_t\|_2} < 0.01$ for $i = k-2, k-1, k$; or $k = K_{\max}$, then $K \leftarrow k$, $\hat{P}_{(j)} \leftarrow [\hat{P}_{(j-1)}\ \hat{P}_{j,\text{new},K}]$ and reset flag $\leftarrow$ detect.
   Else $\hat{P}_t \leftarrow \hat{P}_{t-1}$.

estimates. Thus, if $\frac{|\hat{T}_{t-2} \cap \hat{T}_{t-1}|}{|\hat{T}_{t-2}|} > 0.5$, then we solve (4.5) with $\lambda = \frac{|\hat{T}_{t-2} \backslash \hat{T}_{t-1}|}{|\hat{T}_{t-1}|}$, else we solve it with $\lambda = 1$.

4.3 Improved support estimation

A simple way to estimate the support is by thresholding the solution of (4.5). This can be improved by using the Add-LS-Del procedure for support and signal value estimation [96]. We proceed as follows. First we compute the set $\hat{T}_{t,\text{add}}$ by thresholding on $\hat{S}_{t,cs}$ in order to retain its k largest magnitude entries. We then compute a LS estimate of S_t on $\hat{T}_{t,\text{add}}$ while setting it to zero everywhere else. As explained earlier, because of the LS step, $\hat{S}_{\text{t,add}}$ is a less biased estimate of S_t than $\hat{S}_{t,cs}$. We let $k = 1.4|\hat{T}_{t-1}|$ to allow for a small increase in the support size from $t-1$ to t. A larger value of k also makes it more likely that elements of the set $(T_t \setminus \hat{T}_{t-1})$ are detected into the support estimate[1].

The final estimate of the support, $\hat{T}_t$, is obtained by thresholding on $\hat{S}_{\text{t,add}}$ using a threshold ω. If ω is appropriately chosen, this step helps to delete some of the extra elements from $\hat{T}_{\text{add}}$ and this ensures that the size of $\hat{T}_t$ does not keep increasing (unless the object's size is increasing). An LS estimate computed on $\hat{T}_t$ gives us the final estimate of S_t, i.e. $\hat{S}_t = \text{LS}(y_t, A, \hat{T}_t)$. We use $\omega = \sqrt{\|M_t\|^2/n}$ except in situations where $\|S_t\| \ll \|L_t\|$ - in this case we use $\omega = 0.25\sqrt{\|M_t\|^2/n}$. An alternate approach is to let ω be proportional to the noise magnitude seen by the ℓ_1 step, i.e. to let $\omega = q\|\hat{\beta}_t\|_\infty$, however this approach required different values of q for different experiments (it is not possible to specify one q that works for all experiments).

The complete algorithm with all the above steps is summarized in Algorithm 1.

4.4 Simplifying subspace update: simple recursive PCA

Even the practical version of p-PCA needs to set $K_{\min}$ and $K_{\max}$ besides also setting b and α. Thus, we also experiment with using PCA to replace p-PCA (it is difficult to prove a performance

[1] Due to the larger weight on the $\|x_{(\hat{T}^c_{t-1})}\|_1$ term as compared to that on the $\|x_{(\hat{T}_{t-1})}\|_1$ term, the solution of (4.5) is biased towards zero on $\hat{T}^c_{t-1}$ and thus the solution values along $(T_t \setminus \hat{T}_{t-1})$ are smaller than the true ones.

Algorithm 2 $[\hat{P}, \hat{\Sigma}] = \text{inc-SVD}(\hat{P}, \hat{\Sigma}, D)$

1. set $D_{\|,proj} \leftarrow \hat{P}'D$ and $D_\perp \leftarrow (I - \hat{P}\hat{P}')D$
2. compute QR decomposition of $D_\perp$, i.e. $D_\perp \stackrel{QR}{=} JK$ (here J is a *basis matrix* and K is an upper triangular matrix)
3. compute the SVD: $\begin{bmatrix} \hat{\Sigma} & D_{\|,proj} \\ 0 & K \end{bmatrix} \stackrel{SVD}{=} \tilde{P}\tilde{\Sigma}\tilde{V}'$
4. update $\hat{P} \leftarrow [\hat{P}\ J]\tilde{P}$ and $\hat{\Sigma} \leftarrow \tilde{\Sigma}$

Note: As explained in [3], due to numerical errors, step 4 done too often can eventually result in $\hat{P}$ no longer being a basis matrix. This typically occurs when one tries to use inc-SVD at every time t, i.e. when D is a column vector. This can be addressed using the modified Gram Schmidt re-orthonormalization procedure whenever loss of orthogonality is detected [3].

guarantee with PCA but that does not necessarily mean that the algorithm itself will not work). The simplest way to do this is to compute the top $\hat{r}$ left singular vectors of $[\hat{L}_1, \hat{L}_2, \ldots \hat{L}_t]$ either at each time t or every α frames. While this is simple, its complexity will keep increasing with time t which is not desirable. Even if we use the last d frames instead of all past frames, d will still need to be large compared to $\hat{r}$ to get an accurate estimate. To address this issue, we can use the recursive PCA (incremental SVD) algorithm given in Algorithm 2. We give the complete algorithm that uses this and a rank $\hat{r}$ truncation step every d frames (motivated by [3]) in Algorithm 3.

4.5 Compressive Measurements: Recovering S_t

Consider the problem of recovering S_t from

$$M_t := AS_t + BL_t$$

when A and B are $m \times n$ and $m \times n_2$ matrices, S_t is an n length vector and L_t is an n_2 length vector. In general m can be larger, equal or smaller than n or n_2. In the compressive measurements' case, $m < n$. To specify the assumptions needed in this case, we need to define the basis matrix for range$(BP_{(j)})$ and we need to define a generalization of the denseness coefficient.

Definition 4.5.1 *Let* $Q_j := basis(BP_{(j)})$ *and let* $Q_{j,new} := basis((I - Q_{j-1}Q_{j-1}')BP_{(j)})$.

Algorithm 3 Practical ReProCS-Recursive-PCA

Input: M_t; **Output:** $\hat{T}_t$, $\hat{S}_t$, $\hat{L}_t$; **Parameters:** q, b, α, set $\alpha = 20$ in all experiments, set q, b as explained in Algorithm 1.

Initialization: $[\hat{P}_0, \Sigma_0] \leftarrow$ approx-basis($[M_1, \dots M_{t_{\text{train}}}], b\%$), $\hat{r} \leftarrow \text{rank}(\hat{P}_0)$, $d \leftarrow 3\hat{r}$, initialize $\hat{P}_{tmp} \leftarrow \hat{P}_0$, $\hat{\Sigma}_{tmp} \leftarrow \Sigma_0$, $\hat{P}_{(t_{\text{train}})} \leftarrow \hat{P}_0$ and $\hat{T}_t \leftarrow [.]$. **For** $t > t_{\text{train}}$ **do**

1. Perpendicular Projection: do as in Algorithm 1.
2. Sparse Recovery: do as in Algorithm 1.
3. Estimate L_t: do as in Algorithm 1.
4. Update $\hat{P}_t$: recursive PCA
   (a) If $\text{mod}(t - t_{\text{train}}, \alpha) = 0$,
      i. $[\hat{P}_{tmp}, \hat{\Sigma}_{tmp}] \leftarrow$ inc-SVD$(\hat{P}_{tmp}, \hat{\Sigma}_{tmp}, [\hat{L}_{t-\alpha+1}, \dots \hat{L}_t])$ where inc-SVD is given in Algorithm 2.
      ii. $\hat{P}_t \leftarrow (\hat{P}_{tmp})_{1:\hat{r}}$

   Else $\hat{P}_t \leftarrow \hat{P}_{t-1}$.

   (b) If $\text{mod}(t - t_{\text{train}}, d) = 0$,
      i. $\hat{P}_{tmp} \leftarrow (\hat{P}_{tmp})_{1:\hat{r}}$ and $\hat{\Sigma}_{tmp} \leftarrow (\hat{\Sigma}_{tmp})_{1:\hat{r},1:\hat{r}}$

Definition 4.5.2 *For a matrix or a vector M, define*

$$\kappa_{s,A}(M) = \kappa_{s,A}(range(M)) := \max_{|T| \le s} \|A_T{}' basis(M)\|_2 \tag{4.6}$$

where $\|.\|_2$ is the vector or matrix 2-norm. This quantifies the incoherence between the subspace spanned by any set of s columns of A and the range of M.

We assume the following.

1. L_t and S_t satisfy the assumptions of Sec 3.1, 3.3.
2. The matrix A satisfies the restricted isometry property [7], i.e. $\delta_s(A) \le \delta_* \ll 1$.
3. The denseness assumption is replaced by: $\kappa_{2s,A}(Q_j) \le \kappa_*$, $\kappa_{2s,A}(Q_{j,\text{new}}) \le \kappa_{\text{new}} < \kappa_*$ for a κ_* that is small compared to one. Notice that this depends on the L_t's and on the matrices A and B.

Assume that we are given an initial training sequence that satisfies $M_t = BL_t$ for $t = 1, 2, \dots t_{\text{train}}$. The goal is to recover S_t at each time t. It is not possible to recover L_t unless B is time-varying (this case is studied in [111]). In many imaging applications, e.g. undersampled fMRI or single-pixel video imaging, $B = A$ ($B = A$ is a partial Fourier matrix for MRI and is a random Gaussian or Rademacher matrix for single-pixel imaging). On the other hand, if L_t is large but low-dimensional sensor noise, then $B = I$ (identity matrix), while A is the measurement matrix.

Let $\tilde{L}_t := BL_t$. It is easy to see that if L_t lies in a slowly changing low-dimensional subspace, the same is also true for the sequence $\tilde{L}_t$. Consider the problem of recovering S_t from $M_t := AS_t + \tilde{L}_t$ when an initial training sequence $M_t := \tilde{L}_t$ for $t = 1, 2, \dots t_{\text{train}}$ is available. Using this sequence, it is possible to estimate its approximate basis $\hat{Q}_0$ as explained earlier. If we then project M_t into the subspace orthogonal to range$(\hat{Q}_0)$, the resulting vector $y_t := \Phi_t M_t$ satisfies

$$y_t = (\Phi_t A)S_t + \beta_t$$

where $\beta_t = \Phi_t BL_t$ is small noise for the same reasons explained earlier. Thus, one can still recover S_t from y_t by ℓ_1 or weighted ℓ_1 minimization followed by support recovery and LS. Then, $\tilde{L}_t$ gets recovered as $\hat{\tilde{L}}_t \leftarrow M_t - A\hat{S}_t$ and this is used for updating its subspace estimate. We summarize the resulting algorithm in Algorithm 4. This is being analyzed in ongoing work [57].

The following lemma explains why some of the extra assumptions are needed for this case.

Lemma 4.5.3 *[57] For a* basis matrix*, Q,*

$$\delta_s((I - QQ')A) \le \kappa_{s,A}(Q)^2 + \delta_s(A)$$

Using the above lemma with $Q \equiv Q_j$, it is clear that incoherence of Q_j w.r.t. any set of $2s$ columns of A along with RIP of A ensures that any s sparse vector x can be recovered from $y := (I - Q_j Q_j')Ax$ by ℓ_1 minimization. In compressive ReProCS, the measurement matrix uses $\hat{Q}_j$ instead of Q_j and also involves small noise. With more work, these arguments can be extended to this case as well [see [57]].

Algorithm 4 Compressive ReProCS

Use Algorithm 1 or 3 with the following changes.

- Replace Φ_t in step 2 by $\Phi_t A$.
- Replace step 3 by $\hat{\tilde{L}}_t \leftarrow M_t - A\hat{S}_t$.
- Use $\hat{\tilde{L}}_t$ in place of $\hat{L}_t$ and $\hat{Q}$ in place of $\hat{P}$ everywhere.

CHAPTER 5. SIMULATION RESULTS

5.1 Model Verification

Firstly, we introduce the methods we used to verify the three basic model assumptions.

5.1.1 Low-dimensional and slow subspace change assumption

We used two background image sequence datasets. The first was a video of lake water motion. For this sequence, $n = 6480$ and the number of images were 1500. The second was an indoor video of window curtains moving due to the wind. There was also some lighting variation. The latter part of this sequence also contains a foreground (various persons coming in, writing on the board and leaving). For this sequence, the image size was $n = 5120$ and the number of background-only images were 1755. Both sequences are posted at http://www.ece.iastate.edu/~hanguo/PracReProCS.html.

First note that any given background image sequence will never be exactly low-dimensional, but only be approximately so. Secondly, in practical data, the subspace does not just change as simply as in the model of Sec 3.1. Typically there are some changes to the subspace at every time t. Moreover, with just one training sequence of a given type, it is not possible to estimate the covariance matrix of L_t at each t and thus one cannot detect the subspace change times. The only thing one can do is to assume that there may be a change every τ frames, and that during these τ frames the L_t's are stationary and ergodic; estimate the covariance matrix of L_t for this period using a time average; compute its eigenvectors corresponding to $b\%$ energy (or equivalently compute the $b\%$ approximate basis of $[L_{t-\tau+1}, \dots L_t]$) and use these as $P_{(j)}$. These can be used to test our assumptions.

Testing for slow subspace change can be done in various ways. In [74, Fig 6], we do this after low-rankifying the video data first. This helps to very clearly demonstrate slow subspace change,

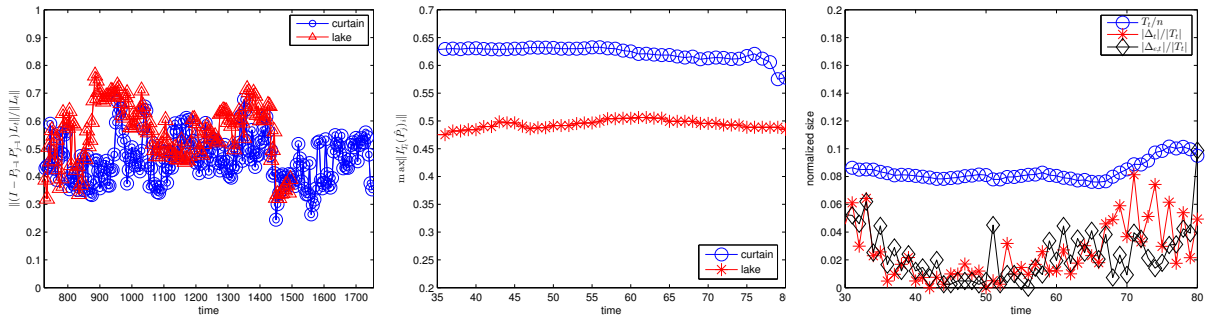
but then it is not checking what we really do on real video data. In this work, we proceed without low-rankifying the data. We let $t_0 = 0$ and $t_j = t_0 + j\tau$ with $\tau = 725$. Let L_t denote the mean subtracted background image sequence, i.e. $L_t = B_t - \mu$ where $\mu = (1/t_1)\sum_{t=0}^{t_1} B_t$. We computed $P_{(j)}$ as $P_{(j)} = \text{approx-basis}([L_{t_j}, ...L_{t_{j+1}-1}], 95\%)$. We observed that $\text{rank}(P_{(j)}) \leq 38$ for curtain sequence, while $\text{rank}(P_{(j)}) \leq 33$ for lake sequence. In other words, 95% of the energy is contained in only 38 or lesser directions in either case, i.e. both sequences are approximately low-dimensional. Notice that the dimension of the matrix $[L_{t_j}, ...L_{t_{j+1}-1}]$ is $n \times \tau$ and $\min(n, \tau) = \tau = 725$ is much larger than 38. To test for slow subspace change, in Fig. 5.1a, we plot $\|(I - P_{(j-1)}P'_{(j-1)})L_t\|_2/\|L_t\|_2$ when $t \in [t_j, t_{j+1})$. Notice that, after every change time ($t_j = 725, 1450$), this quantity is initially small for the first 100-150 frames and then increases gradually. It later decreases also but that is allowed (all we need is that it be small initially and increase slowly).

5.1.2 Denseness assumption

Exactly verifying the denseness assumption is impossible since computing $\kappa_s(.)$ has exponential complexity (one needs to check all sets T of size s). Instead, to get some idea if it holds even just for T replaced by T_t, in Fig. 5.1b, we plot $\max_i \|I_{T_t}{}'(P_{(j)})_i\|_2$ where T_t is the true or estimated support of S_t at time t. For the lake sequence, T_t is simulated and hence known. For the curtain sequence, we select a part of the sequence in which the person is wearing a black shirt (against a white curtains' background). This part corresponds to $t = 35$ to $t = 80$. For this part, ReProCS returns a very accurate estimate of T_t, and we use this estimated support as a proxy for the true support T_t.

5.1.3 Support size, support change and slow support change

For real video sequences, it is not possible to get the true foreground support. Thus we used $\hat{T}_t$ for the part of the curtain sequence described above in Sec 5.1.2 as a proxy for T_t. We plot the support size normalized by the image size $|T_t|/n$, and we plot the number of additions and removals normalized by the support size, i.e. $|\Delta_t|/|T_t|$ and $|\Delta_{e,t}|/|T_t|$ in Fig. 5.1c. Notice from the



(a) (b) (c)

Figure 5.2: (a) Verification of slow subspace change assumption. (b) Verification of denseness assumption. (c) Verification of small support size, small support change

figure that the support size is at most 10.2% of the image size. Notice also that at least at every 3 frames, there is at least a 1% support change. Thus there is some support change every few frames, thus exposing the part of the background behind the foreground. Finally notice that the maximum number of support changes is only 9.9% of the support size, i.e. slow support change holds for this piece.

5.2 Experimental results

In this section, we show comparisons of prac-ReProCS with other batch and recursive algorithms for robust PCA. For implementing the ℓ_1 or weighted ℓ_1 minimizations, we used the YALL1 ℓ_1 minimization toolbox [107], its code is available at http://yall1.blogs.rice.edu/.

Code and data for our algorithms and for all experiments given below is available at http://www.ece.iastate.edu/~hanguo/ReProCS_demo.rar.

5.2.1 Simulated data

In this experiment, the measurement at time t, $M_t := L_t + S_t$, is an $n \times 1$ vector with $n = 100$. We generated L_t using the autoregressive model described in [71] with auto-regression parameter 0.1, and the decay parameter $f_d = 0.1$. The covariance of a direction decayed to zero before being removed. There was one change time t_1. For $t < t_1$, $P_t = P_0$ was a rank $r_0 = 20$ matrix and

$Cov(a_t)$ was a diagonal matrix with entries $10^4, 0.7079 \times 10^4, 0.7079^2 \times 10^4, \cdots, 14.13$. At $t = t_1$, $c = c_{1,\text{new}} = 2$ new directions, $P_{1,\text{new}}$, got added with $Cov(a_{t,\text{new}})$ being diagonal with entries 60 and 50. Also, the variance along two existing directions started to decay to zero exponentially. We used $t_{\text{train}} = 2000$ and $t_1 = t_{\text{train}} + 5$. The matrix $[P_0\ P_{1,\text{new}}]$ was generated as the first 22 columns of an $n \times n$ random orthonormal matrix (generated by first generating an $n \times n$ matrix random Gaussian matrix and then orthonormalizing it). For $1 \le t \le t_{\text{train}}$, $S_t = 0$ and hence $M_t = L_t$. For $t > t_{\text{train}}$, the support set, T_t, was generated in a correlated fashion: S_t contained one block of size 9 or 27 (small and large support size cases). The block stayed static with probability 0.8 and move up or down by one pixel with probability 0.1 each independently at each time. Thus the support sets were highly correlated. The magnitude of the nonzero elements of S_t is fixed at either 100 (large) or 10 (small).

For the small magnitude S_t case, $\|L_t\|_2$ ranged from 150 to 250 while $\|S_t\|_2$ was equal to 30 and 52, i.e. in this case $\|S_t\|_2 \ll \|L_t\|_2$. For the large magnitude case, $\|S_t\|_2$ was 300 and 520. We implemented ReProCS (Algorithm 1) with $b = 99.99$ since this data is exactly low-rank. We used $q = 0.25$ for the small magnitude S_t case and $q = 1$ for the other case. We compared with three recursive robust PCA methods – incremental robust subspace learning (iRSL) [51] and adapted (outlier-detection enabled) incremental SVD (adapted-iSVD) [3] and GRASTA [33] – and with two batch methods – Principal Components' Pursuit (PCP) [10] [1] and robust subspace learning (RSL)[2] [94]. Results are shown in Table 5.1.

From these experiments, we can conclude that ReProCS is able to successfully recover both small magnitude and fairly large support-sized S_t's; iRSL has very bad performance in both cases; RSL, PCP and GRASTA work to some extent in certain cases, though not as well as ReProCS. ReProCS operates by first approximately nullifying L_t, i.e. computing y_t as in (4.2), and then recovering S_t by exploiting its sparsity. iRSL and RSL also compute y_t the same way, but they directly use y_t to detect or soft-detect (and down-weight) the support of S_t by thresholding. Recall that y_t can be

[1] We use the Accelerated Proximal Gradient algorithm[54] and Inexact ALM algorithm [53] (designed for large scale problems) to solve PCP (2.1). The code is available at http://perception.csl.uiuc.edu/matrix-rank/sample_code.html.

[2] The code of RSL is available at http://www.salleurl.edu/ ftorre/papers/rpca/rpca.zip.

Table 5.1: Comparison of reconstruction errors of different algorithms for simulated data. Here, $|T_t|/n$ is the sparsity ratio of S_t, $\mathbb{E}[.]$ denotes the Monte Carlo average computed over 100 realizations and $\|.\|_F$ is the Frobenius norm of a matrix. Also, $S = [S_1, S_2, \dots S_{t_{\max}}]$ and $\hat{S}$ is its estimate; $(O_t)_i = (M_t)_i$ if $i \in T_t$ and $(O_t)_i = 0$ otherwise and $\hat{O}_t$ is defined similarly with the estimates. O and $\hat{O}$ are the corresponding matrices. We show error for O for iRSL and adapted-iSVD since these algorithms can only return an estimate of the outlier support T_t; they do not return the background estimate.

(a) $(S_t)_i = 100$ for $i \in T_t$ and $(S_t)_i = 0$ for $i \in T_t^c$						
	$\mathbb{E}\|S - \hat{S}\|_F^2/\mathbb{E}\|S\|_F^2$				$\mathbb{E}\|O - \hat{O}\|_F^2/\mathbb{E}\|O\|_F^2$	
$\|T_t\|/n$	ReProCS-pPCA	RSL	PCP	GRASTA	adapted-iSVD	iRSL
9%	1.52×10^{-4}	0.0580	0.0021	3.75×10^{-4}	0.0283	0.9105
27%	1.90×10^{-4}	0.0198	0.6852	0.1043	0.0637	0.9058
(b) $(S_t)_i = 10$ for $i \in T_t$ and $(S_t)_i = 0$ for $i \in T_t^c$						
	$\mathbb{E}\|S - \hat{S}\|_F^2/\mathbb{E}\|S\|_F^2$				$\mathbb{E}\|O - \hat{O}\|_F^2/\mathbb{E}\|O\|_F^2$	
$\|T_t\|/n$	ReProCS-pPCA	RSL	PCP	GRASTA	adapted-iSVD	iRSL
9%	0.0344	8.7247	0.2120	0.1390	0.2346	0.9739
27%	0.0668	3.3166	0.6456	0.1275	0.3509	0.9778

rewritten as $y_t = S_t + (-\hat{P}_{t-1}\hat{P}_{t-1}' S_t) + \beta_t$. As the support size of S_t increases, the interference due to $(-\hat{P}_{t-1}\hat{P}_{t-1}' S_t)$ becomes larger, resulting in wrong estimates of S_t. For the same reason, direct thresholding is also difficult when some entries of S_t are small while others are not. Adapted-iSVD is our adaptation of iSVD [3] in which we use the approach of iRSL described above to provide the outlier locations to iSVD (iSVD is an algorithm for recursive PCA with missing entries or what can be called recursive low-rank matrix completion). It fills in the corrupted locations of L_t by imposing that L_t lies in range$(\hat{P}_{t-1})$. We used a threshold of $0.5 \min_{i \in T_t} |(S_t)_i|$ for both iRSL and adapted-iSVD (we also tried various other options for thresholds but with similar results). Since adapted-iSVD and iRSL are recursive methods, a wrong $\hat{S}_t$, in turn, results in wrong subspace updates, thus also causing β_t to become large and finally causing the error to blow up.

RSL works to some extent for larger support size of S_t's but fails when the magnitude of the nonzero S_t's is small. PCP fails since the support sets are generated in a highly correlated fashion and the support sizes are large (resulting in the matrix $\mathcal{S}_t$ being quite rank deficient also). GRASTA [33] is a recent recursive method from 2012. It was implemented using code posted at https://sites.google.com/site/hejunzz/grasta. We tried two versions of GRASTA: the demo code as it is and the demo code modified so that it used as much information as ReProCS used

(i.e. we used all available frames for training instead of just 100; we used all measurements instead of just 20% randomly selected pixels; and we used $\hat{r}$ returned by ReProCS as the rank input instead of using rank=5 always). In this paper, we show the latter case. Both experiments are shown on our supplementary material page http://www.ece.iastate.edu/~hanguo/PracReProCS.html.

5.2.2 Partly Simulated Video: Lake video with simulated foreground

In the comparisons shown next, we only compare with PCP, RSL and GRASTA. To address a reviewer comment, we also compare with the batch algorithm of [61, 59] (referred to as MG in the figures) implemented using code provided by the authors. There was not enough information in the papers or in the code to successfully implement the recursive algorithm.

We implemented ReProCS (Algorithm 1 and Algorithm 3) with $b = 95$ since the videos are only approximately low-rank and we used $q = 1$ since the magnitude of S_t is not small compared to that of L_t. The performance of both ReProCS-pPCA (Algorithm 1) and ReProCS-recursive-PCA (Algorithm 3) was very similar. Results with using the latter are shown in Fig. 5.9.

We used the lake sequence described earlier to serve as a real background sequence. Foreground consisting of a rectangular moving object was overlaid on top of it using (3.2). The use of a real background sequence allows us to evaluate performance for data that only approximately satisfies the low-dimensional and slow subspace change assumptions. The use of the simulated foreground allows us to control its intensity so that the resulting S_t is small or of the same order as L_t (making it a difficult sequence), see Fig. 5.3b.

The foreground F_t was generated as follows. For $1 \le t \le t_{\text{train}}$, $F_t = 0$. For $t > t_{\text{train}}$, F_t consists of a 45×25 moving block whose centroid moves horizontally according to a constant velocity model with small random acceleration [70, Example V.B.2]. To be precise, let p_t be the horizontal location of the block's centroid at time t, let v_t denote its horizontal velocity. Then $g_t := \begin{bmatrix} p_t \\ v_t \end{bmatrix}$ satisfies $g_t = G g_{t-1} + \begin{bmatrix} 0 \\ n_t \end{bmatrix}$ where $G := \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ and n_t is a zero mean truncated Gaussian with variance Q and with $-2\sqrt{Q} < |n_t| < 2\sqrt{Q}$. The nonzero pixels' intensity is i.i.d. over time and space

Table 5.2: Comparison of speed of different algorithms. Experiments were done on a 64 bit Windows 8 laptop with 2.40GHz i7 CPU and 8G RAM. Sequence length refers to the length of sequence for training plus the length of sequence for separation. For ReProCS and GRASTA, the time is shown as training time + recovery time.

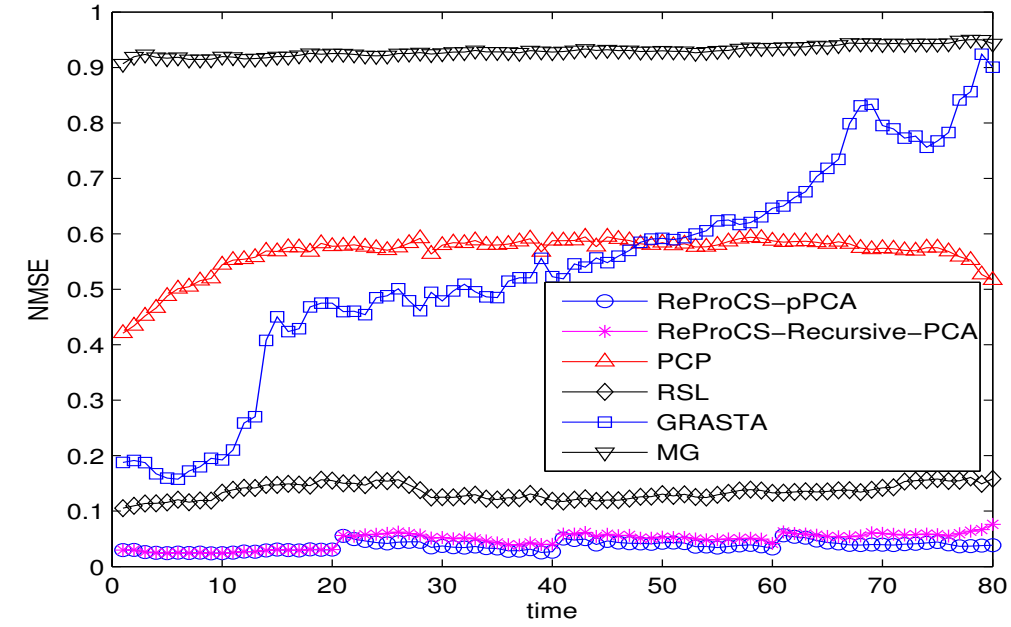
DataSet	Image Size	Sequence Length	ReProCS-pPCA	ReProCS-Recursive-PCA	PCP	RSL	GRASTA
Lake	72×90	$1420 + 80$	$2.99 + 19.97$ sec	$2.99 + 19.43$ sec	245.03 sec	213.36 sec	$39.47 + 0.42$ sec
Curtain	64×80	$1755 + 1209$	$4.37 + 159.02$ sec	$4.37 + 157.21$ sec	1079.59 sec	643.98 sec	$40.01 + 5.13$ sec
Person	120×160	$200 + 52$	$0.46 + 42.43$ sec	$0.46 + 41.91$ sec	27.72 sec	121.31 sec	$13.80 + 0.64$ sec

and distributed as uniform(b_1, b_2), i.e. $(F_t)_i \sim$ uniform(b_1, b_2) for $i \in T_t$. In our experiments, we generated the data with $t_{\text{train}} = 1420$, $b_1 = 170$, $b_2 = 230$, $p_{t_0+1} = 27$, $v_{t_0+1} = 0.5$ and $Q = 0.02$. With these values of b_1, b_2, as can be seen from Fig. 5.3b, $\|S_t\|_2$ is roughly equal or smaller than $\|L_t\|_2$ making it a difficult sequence. Since it is not much smaller, ReProCS used $q = 1$; since background data is approximately low-rank it used $b = 95$.

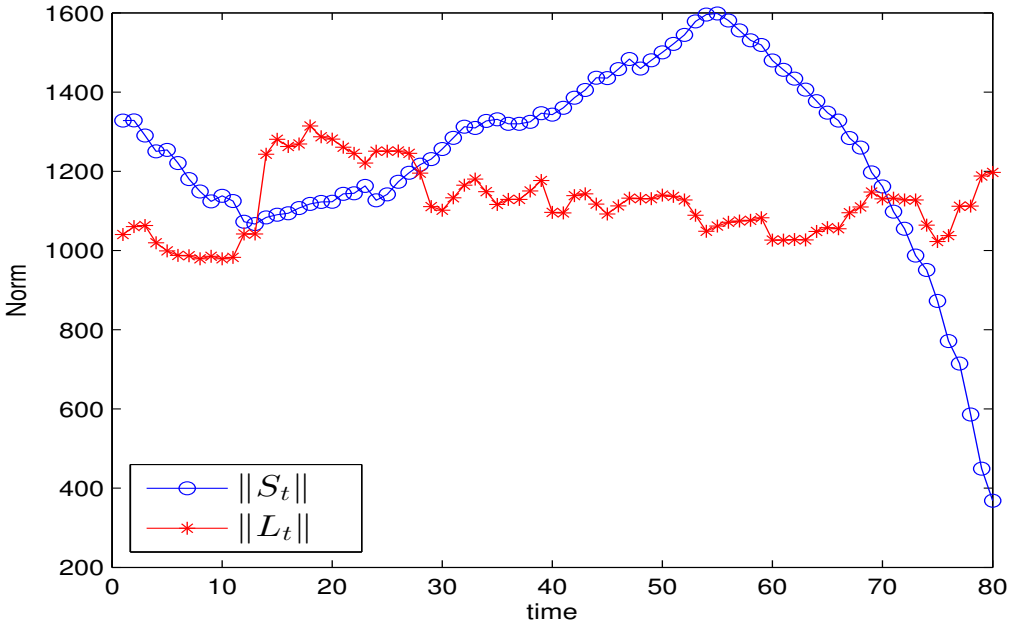
We generated 50 realizations of the video sequence using these parameters and compared all the algorithms to estimate S_t, L_t and then the foreground and the background sequences. We show comparisons of the normalized mean squared error (NMSE) in recovering S_t in Fig. 5.3a. Visual comparisons of both foreground and background recovery for one realization are shown in Fig. 5.5. The recovered foreground image is shown as a white-black image showing the foreground support: pixels in the support estimate are white. PCP gives large error for this sequence since the object moves in a highly correlated fashion and occupies a large part of the image. GRASTA also does not work. RSL is able to recover a large part of the object correctly, however it also recovers many more extras than ReProCS. The reason is that the magnitude of the nonzero entries of S_t is quite small (recall that $(S_t)_i = (F_t - B_t)_i$ for $i \in T_t$) and is such that $\|L_t\|_2$ is about as large as $\|S_t\|_2$ or sometimes larger (see Fig. 5.3b).

5.2.3 Real video sequences

Next we show comparisons on two real video sequences. These are originally taken from http://perception.i2r.a-star.edu.sg/bk_model/bk_index.html and http://research.microsoft.com/en-us/um/people/jckrumm/wallflower/testimages.htm, respectively. The first is the cur-



(a)



(b)

Figure 5.4: Experiments on partly simulated video. (a) Normalized mean squared error in recovering S_t for realizations. (b) Comparison of $\|S_t\|_2$ and $\|L_t\|$ for one realization. MG refers to the batch algorithm of [61, 59] implemented using code provided by the authors. There was not enough information in the papers or in the code to successfully implement the recursive algorithm.

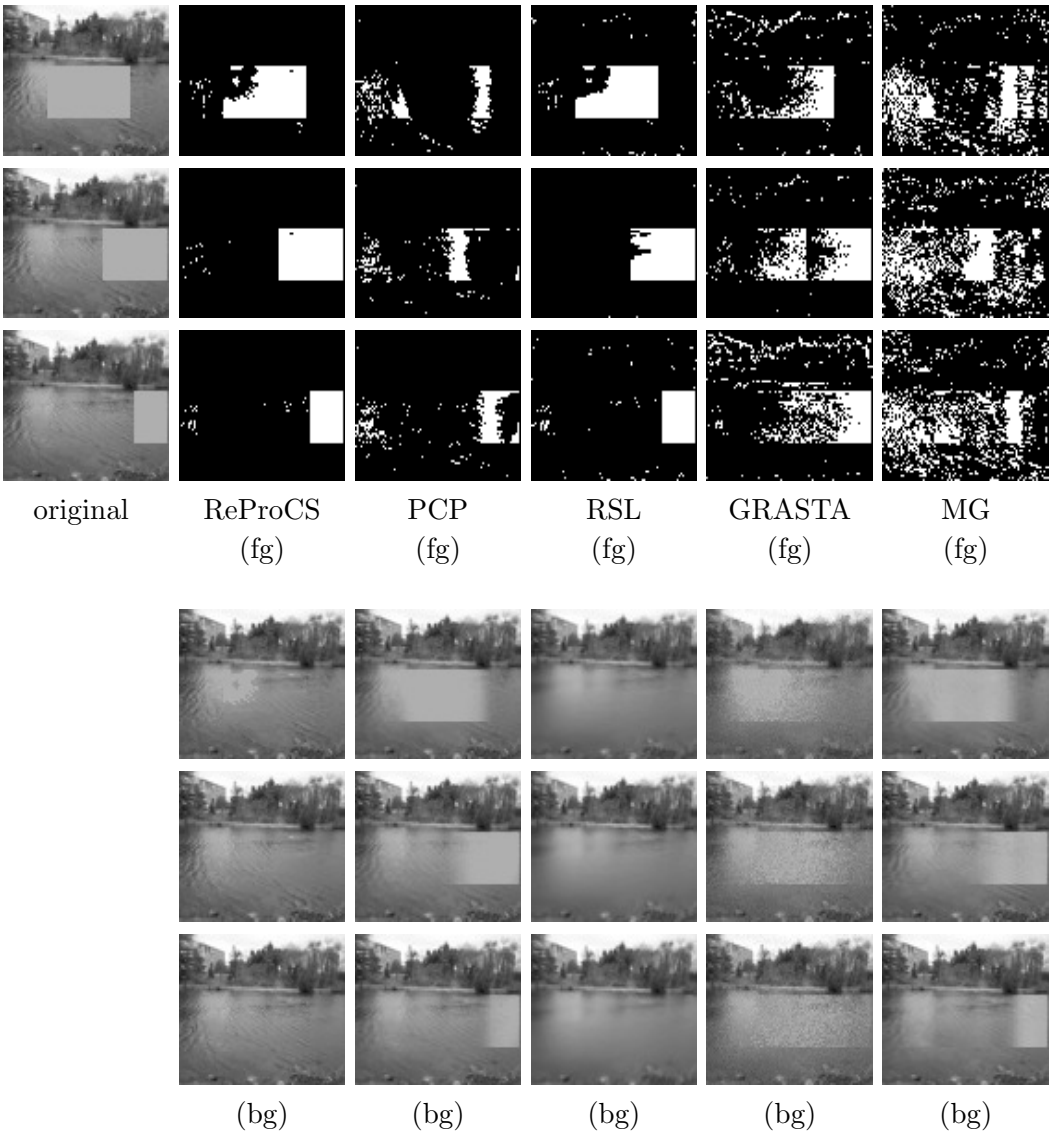


Figure 5.5: Original video at $t = t_{\text{train}} + 30, 60, 70$ and its foreground (fg) and background (bg) layer recovery results using ReProCS (ReProCS-pCA) and other algorithms. MG refers to the batch algorithm of [61, 59] implemented using code provided by the authors. There was not enough information in the papers or in the code to successfully implement the recursive algorithm. For fg, we only show the fg support in white for ease of display.

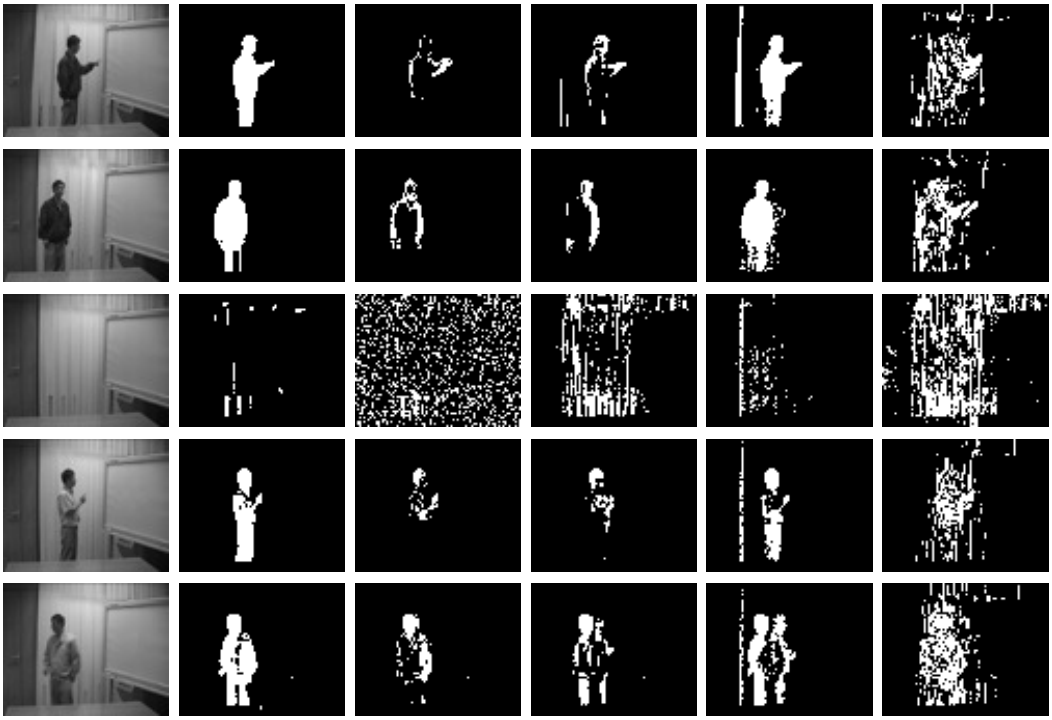


Figure 5.6: Original video sequence at $t = t_{\text{train}} + 60, 120, 199, 475, 1148$ and its foreground (fg) and background (bg) layer recovery results using ReProCS (ReProCS-pCA) and other algorithms. For fg, we only show the fg support in white for ease of display.

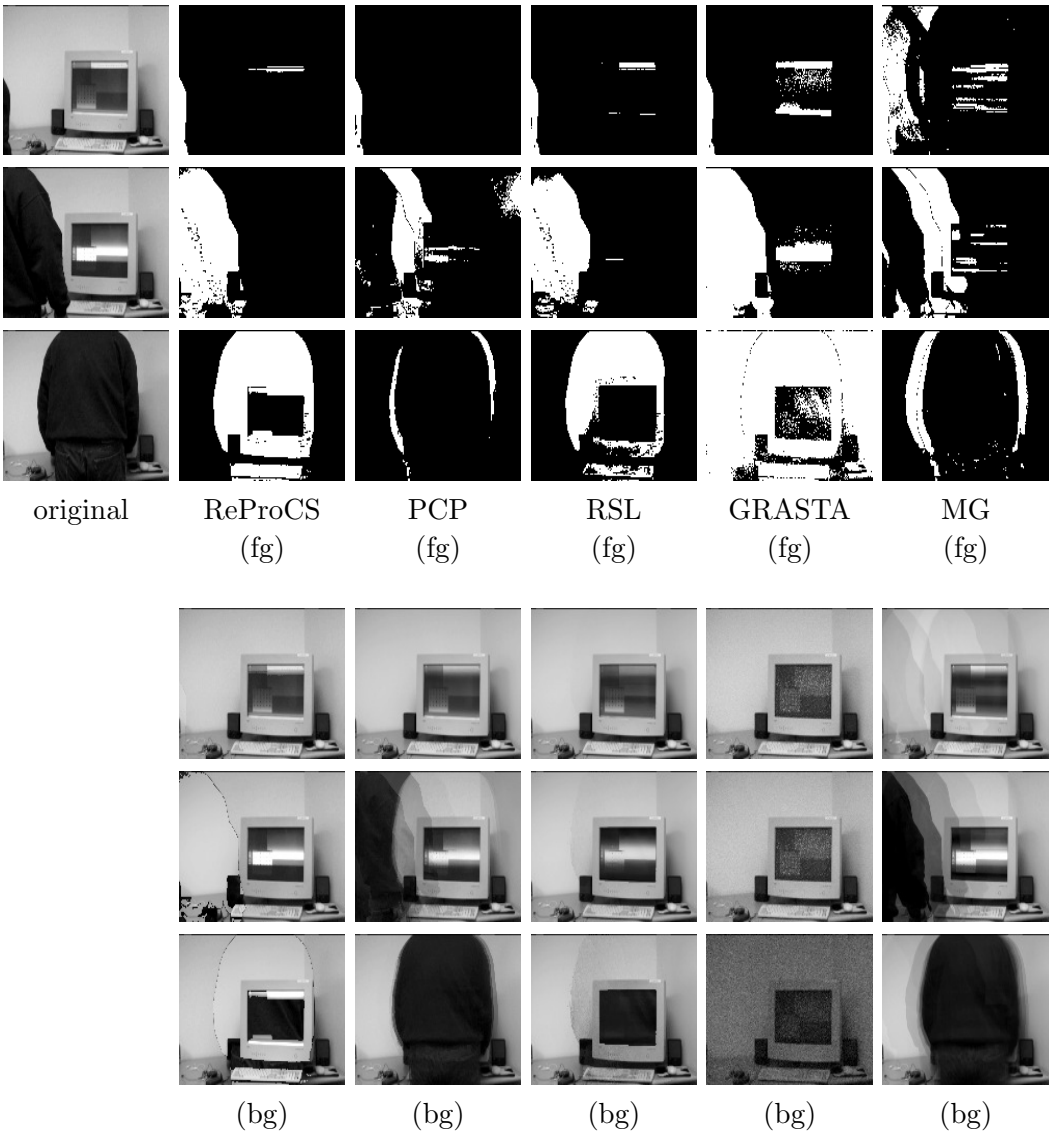


Figure 5.7: Original video sequence at $t = t_{\text{train}} + 42, 44, 52$ and its foreground (fg) and background (bg) layer recovery results using ReProCS (ReProCS-pCA) and other algorithms. For fg, we only show the fg support in white for ease of display.

(a) $t = 30, 60, 70$ (b) $t = 60, 120, 475$ (c) $t = 42, 44, 52$

Figure 5.9: Foreground layer estimated by ReProCS-Recursive-PCA for the lake, curtain and person videos shown in Figs 5.5, 5.6 and 5.7. As can be seen the recovery performance is very similar to that of ReProCS-pPCA (Algorithm 1).

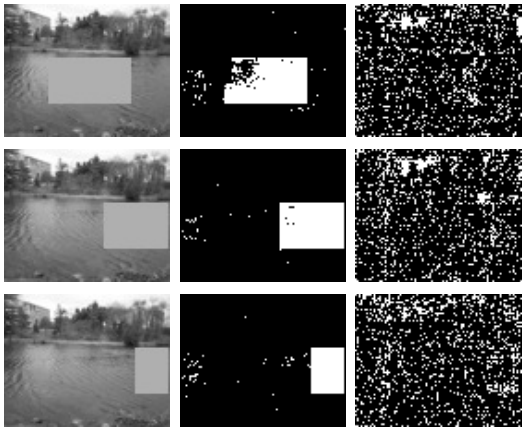


Figure 5.10: Original video frames at $t = t_{\text{train}} + 30, 60, 70$ and foreground layer recovery by ReProCS and SparCS.

tain sequence described earlier. For $t > 1755$, in the foreground, a person with a black shirt walks in, writes on the board and then walk out, then a second person with a white shirt does the same and then a third person with a white shirt does the same. This video is challenging because (i) the white shirt color and the curtains' color is quite similar, making the corresponding S_t small in magnitude; and (ii) because the background variations are quite large while the foreground person moves slowly. As can be seen from Fig. 5.6, ReProCS's performance is significantly better than that of the other algorithms for both foreground and background recovery. This is most easily seen from the recovered background images. One or more frames of the background recovered by PCP, RSL and GRASTA contains the person, while none of the ReProCS ones does.

The second sequence consists of a person entering a room containing a computer monitor that contains a white moving region. Background changes due to lighting variations and due to the computer monitor. The person moving in the foreground occupies a very large part of the image, so this is an example of a sequence in which the use of weighted ℓ_1 is essential (the support size is too large for simple ℓ_1 to work). As can be seen from Fig. 5.7, for most frames, ReProCS is able to recover the person correctly. However, for the last few frames which consist of the person in a white shirt in front of the white part of the screen, the resulting S_t is too small even for ReProCS

to correctly recover. The same is true for the other algorithms. Videos of all above experiments and of a few others are posted at http://www.ece.iastate.edu/~hanguo/PracReProCS.html.

Time Comparisons. The time comparisons are shown in Table 5.2. In terms of speed, GRASTA is the fastest even though its performance is much worse. ReProCS is the second fastest. We expect that ReProCS can be speeded up by using mex files (C/C++ code) for the subspace update step. PCP and RSL are slower because they jointly process the entire image sequence. Moreover, ReProCS and GRASTA have the advantage of being recursive methods, i.e. the foreground/background recovery is available as soon as a new frame appears while PCP or RSL need to wait for the entire image sequence.

5.2.4 Compressive ReProCS: comparisons for simulated video.

We compare compressive ReProCS with SpaRCS [97] which is a batch algorithm for undersampled robust PCA / separation of sparse and low-dimensional parts(its code is downloaded from http://www.ece.rice.edu/~aew2/sparcs.html). No code is available for most of the other compressive batch robust PCA algorithms such as [100, 28]. SpaRCS is a greedy approach that combines ideas from CoSaMP [65] for sparse recovery and ADMiRA [49] for matrix completion. The comparison is done for compressive measurements of the lake sequence with foreground simulated as explained earlier. The matrix $B = A$ is $m \times n$ random Gaussian with $m = 0.7n$. Recall that $n = 6480$. The SpaRCS code required the background data rank and foreground sparsity as inputs. For rank, we used $\hat{r}$ returned by ReProCS, for sparsity we used the true size of the simulated foreground. As can be seen from Fig. 5.10, SpaRCS does not work while compressive ReProCS is able to recover S_t fairly accurately, though of course the errors are larger than in the full sampled case. All experiments shown in [97] are for very slow changing backgrounds and for foregrounds with very small support sizes, while neither is true for our data.

5.3 Conclusion

We designed and evaluated prac-ReProCS which is a practically usable modification of its theoretical counterpart that was studied in earlier work [74, 73, 1]. We showed that prac-ReProCS has excellent performance for both simulated data and for a real application (foreground-background separation in videos) and the performance is better than many of the state-of-the-art algorithms from recent work. Moreover, most of the assumptions used to obtain its guarantees are valid for real videos. Finally we also proposed and evaluated a compressive prac-ReProCS algorithm. In ongoing work, on one end, we are working on performance guarantees for compressive ReProCS [57] and on the other end, we are developing and evaluating a related approach for functional MRI. In fMRI, one is allowed to change the measurement matrix at each time. However if we replace $B = A$ by A_t in Sec 4.5 the compressive ReProCS algorithm does not apply because $A_t L_t$ is not low-dimensional [111].

CHAPTER 6. VIDEO DENOISING WITH PRAC-REPROCS

6.1 Introduction

Video denoising refers to the problem of removing "noise" from a video sequence. Here the term "noise" is used in a broad sense to refer to any corruption or outlier or interference that is not the quantity of interest. In the last few decades there has been a lot of work on video denoising. An important direction for denoising is "grouping and collaborative filtering" approaches which try to search for similar image patches both within an image frame and across nearby frames, followed by collaboratively filtering the noise from the stack of matched patches [4, 22, 23, 26, 58, 14]. One of the most effective methods for image denoising, Block Matching and 3D filtering (BM3D) [22], is from this category of techniques. In BM3D, similar image blocks are stacked in a 3D array followed by applying a noise shrinkage operator in a transform domain (illustration of grouping shown in Figure.6.1). In its video version, VBM3D [21], the method is generalized to video denoising by searching for similar blocks across multiple frames.

Other related works [42, 41] apply batch matrix completion or matrix decomposition on grouped image patches to remove outliers. [15] studies performance bounds for image denoising and shows

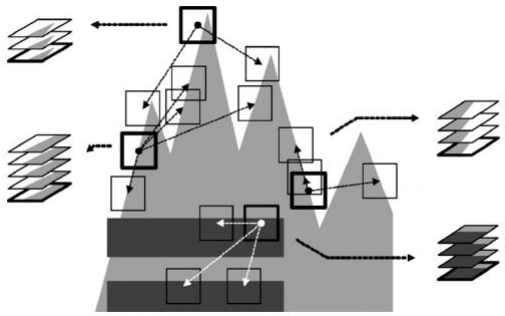


Figure 6.1: Illustration of BM3D: a simple example of grouping in an artificial image, where for each reference block (with thick borders) there exist perfectly similar ones (image from [22]).

that there is still room for improvement. Recent works on video denoising include approaches that use motion compensation algorithms from the video compression literature followed by denoising of similar nearby blocks [55, 31]; and approaches that use wavelet transform based [76, 75, 109] and discrete cosine transform (DCT) based [45] denoising solutions. Very recent video denoising methods include algorithms based on sparsifying transform learning [99, 98]. Another important denoising approach is based on deep learning, e.g., [104, 2, 112] for image denoising and [18] for video denoising.

In this chapter we develop a novel denoising solution framework, that we call *Layering-Denoising (LD)*, for highly noisy or otherwise corrupted videos that are well modeled as the sum of a dense low-rank matrix and a sparse matrix with high enough rank (not too many nonzeroes in any row or column). We refer to these components as the "low-rank layer" and the "sparse layer" of the video. For such videos, we show that the performance of existing state-of-the-art denoisers can be significantly improved (especially in very large noise settings) if the video is first decomposed into the two layers, and the denoiser is applied on each layer separately. After this, depending on the application of interest, either just the denoised low-rank layer, or just the denoised sparse layer can be outputted, or the two denoised layers can be added back and the denoised video outputted. A large class of videos fit in the above category. Many clean videos are slowly changing and these are well modeled as forming a low-rank matrix. Large noise (including large Gaussian noise) can be split into the sum of a small bounded component and a large magnitude sparse component. We explain this fact in detail in Sec. 6.2. Thus, slowly changing videos corrupted by either large noise or by salt-and-pepper noise (which, by definition, is sparse), are well modeled as being low-rank+sparse. Moreover, many other videos consist of slow-changing backgrounds plus sparse foreground moving objects. Noisy versions of such videos are also correctly modeled as low-rank plus sparse with the sparse layer now consisting of both the foreground and the large components of the noise. A third application is low-light video denoising or "seeing in the dark". *Layering-Denoising* allows one to see the moving objects which are barely visible otherwise.

Figure 6.2: Example of video denoising: detecting invisible object in the dark.

We develop a fast and memory-efficient denoising solution called *ReProCS-based Layering Denoising (ReLD)*. This consists of an online algorithm based on Recursive Projected Compressive Sensing (ReProCS) framework for the layering task and VBM3D [21] for denoising each of the layers. We pick the ReProCS approach because ReProCS-based algorithms are known to provide online (after a short batch initialization), fast, memory-efficient, and highly-robust robust PCA solutions [30, 64].

Example applications. Many videos that require denoising/enhancement can be accurately modeled in the above fashion. All videos referenced below are posted at http://www.ece.iastate.edu/~hanguo/denoise.html.

1. In very low-light videos of moving targets/objects (the moving target is barely visible), the denoising goal is to "see" the barely visible moving targets (sparse). These are hard to see because they are corrupted by slowly-changing background images (shown in Figure.6.2).

2. Consider slowly changing videos that are corrupted by salt-and-pepper noise (or other impulsive noise). For these types of videos, the large magnitude part of the noise forms the "sparse layer", while the video-of-interest (slowly-changing in many applications, e.g., waterfall, waving trees, sea water moving, etc) forms the approximate "low-rank layer". The goal is to denoise or extract out the "low-rank layer" (shown in Figure.6.3).

3. Consider slow-changing videos corrupted by very large variance white Gaussian noise. As we explain below, large Gaussian noise can, with high probability, be split into a very sparse noise component plus bounded noise.

Figure 6.3: Example of video denoising: salt-and-pepper noise.

Figure 6.4: Example of video denoising: large Gaussian noise.

4. In videos where foreground objects are also present, the video itself become "low-rank + sparse". In such a scenario, the "sparse layer" that is extracted out will consist of the foreground object and the large magnitude part of the noise. Some examples are the curtain and lobby videos (shown at http://www.ece.iastate.edu/~hanguo/denoise.html). The proposed ReLD algorithm works for these videos if VBM3D applied to the foreground layer video is able to separate out the foreground moving object(s) from the noise.

Moreover, in all these examples, it is valid to argue that the columns of the low-rank matrix lie in a low-dimensional subspace that is either fixed or slowly changing. This is true, for example, when the background consists of moving waters, or the background changes are due to illumination variations. These also result in global (non-sparse) changes.

6.2 Problem formulation

Let M_t denote the image at time t arranged as a 1D vector of length n. We consider denoising for videos in which each image can be split as

$$M_t = L_t + S_t + W_t, \quad t = 1, 2, \ldots, t_{\max}$$

where S_t is a sparse vector, L_t's lie in a fixed or slowly changing low-dimensional subspace of $\mathbb{R}^n$ so that the matrix $\mathcal{L} := [L_1, L_2, \ldots, L_{t_{\max}}]$ is low-rank, and W_t is the residual noise that satisfies $\|W_t\|_\infty \leq b_w$. We use $\mathcal{T}_t$ to denote the support set of S_t, i.e., $\mathcal{T}_t := \text{support}(S_t)$.

In the first example given above, the moving targets' layer is S_t, the slowly-changing dark background is $L_t + W_t$. The layer of interest is S_t. In the second example, the slowly changing video is $L_t + W_t$, while the salt-and-pepper noise is S_t. The layer of interest is L_t. In the third example, the slowly changing video is $L_t + W_{1,t}$ with $W_{1,t}$ being the residual; and, as we explain next, with high probability (whp), white Gaussian noise can be split as $S_t + W_{2,t}$ with $W_{2,t}$ being bounded. In this case, $W_t = W_{1,t} + W_{2,t}$.

Let $\boldsymbol{n}$ denote a Gaussian noise vector in $\mathbb{R}^n$ with zero mean and covariance $\sigma^2 \boldsymbol{I}$. Let $\beta(b) := 2\Phi(b) - 1$ with $\Phi(z)$ being the cumulative distribution function (CDF) of the standard Gaussian distribution. Then, it is not hard to see that $\boldsymbol{n}$ can be split as

$$\boldsymbol{n} = \boldsymbol{s} + \boldsymbol{w}$$

where $\boldsymbol{w}$ is bounded noise with $\|\boldsymbol{w}\|_\infty \leq b_0$ and $\boldsymbol{s}$ is a sparse vector with support size $|\mathcal{T}_t| \approx \left(1 - \beta\left(\frac{b_0}{\sigma}\right)\right) n$ whp. More precisely, with probability at least $1 - 2\exp(-2\epsilon^2 n)$,

$$\left(1 - \beta\left(\frac{b_0}{\sigma}\right) - \epsilon\right) n \leq |\mathcal{T}_t| \leq \left(1 - \beta\left(\frac{b_0}{\sigma}\right) + \epsilon\right) n.$$

In words, whp, $\boldsymbol{s}$ is sparse with support size roughly $(1-\beta)n$ where $\beta = \beta\left(\frac{b_0}{\sigma}\right)$. The above claim is a direct consequence of Hoeffding's inequality for a sum of independent Bernoulli random variables [1].

[1] If p is the probability of $z_i = 1$, then Hoeffding's inequality says that:

$$\Pr((p - \epsilon)n \leq \sum_i z_i \leq (p + \epsilon)n) \geq 1 - 2\exp(-2\varepsilon^2 n)$$

We apply it to the Bernoulli random variables z_i's with z_i defined as $z_i = 1$ if $\{\boldsymbol{s}_i \neq 0\}$ and $z_i = 0$ if $\{\boldsymbol{s}_i = 0\}$. Clearly, $\Pr(z_i = 0) = \Pr(\boldsymbol{s}_i = 0) = \Pr(\boldsymbol{n}_i^2 \leq b_0^2) = \Phi(b_0/\sigma) - \Phi(-b_0/\sigma) = 2\Phi(b_0/\sigma) - 1 = \beta(b_0/\sigma)$.

6.3 ReProCS-based Layering Denoising (ReLD)

We summarize the ReProCS-based Layering Denoising (ReLD) algorithm in Algorithm 5. The approach is explained below.

Initialization. Take $\mathcal{M}_0 = [M_1, M_2, \cdots, M_{t_0}]$ as training data and use PCP [10] to separate it into a sparse matrix $[\hat{S}_1, \hat{S}_2, \cdots, \hat{S}_{t_0}]$ and a low-rank matrix $[\hat{L}_1, \hat{L}_2, \cdots, \hat{L}_{t_0}]$. Compute the top $q\%$ left singular vectors of $[\hat{L}_1, \hat{L}_2, \cdots, \hat{L}_{t_0}]$ denoted by $\hat{P}_0$. Here $q\%$ left singular vectors of a matrix $\mathcal{M}$ refer to the left singular vectors of $\mathcal{M}$ whose corresponding singular values form the smallest set of singular values that contains at least $q\%$ of the total singular values' energy.

Splitting phase. Let $\hat{P}_{t-1}$ be the basis matrix (matrix with orthonormal columns) for the estimated subspace of L_{t-1}. For $t \geq t_0 + 1$, we split M_t into $\hat{S}_t$ and $\hat{L}_t$ using prac-ReProCS [30]. To do this, we first project M_t onto the subspace orthogonal to range$(\hat{P}_{t-1})$ to get the projected measurement vector,

$$y_t := (I - \hat{P}_{t-1}\hat{P}_{t-1}')M_t := \Phi_t M_t. \tag{6.1}$$

Observe that y_t can be expressed as

$$y_t = \Phi_t S_t + \beta_t \text{ where } \beta_t := \Phi_t(L_t + W_t). \tag{6.2}$$

Because of the slow subspace change assumption, the projection nullifies most of the contribution of L_t and hence β_t is small noise. The problem of recovering S_t from y_t becomes a traditional noisy sparse recovery/CS problem and one can use L_1 minimization or any of the greedy or iterative thresholding algorithms to solve it. We denote its solution by $\hat{S}_t$, and obtain $\hat{L}_t$ by simply subtracting $\hat{S}_t$ from M_t.

Denoising phase. We perform VBM3D on $\hat{\mathcal{S}} = [\hat{S}_1, \ldots, \hat{S}_{t_{\max}}]$ and $\hat{\mathcal{L}} = [\hat{L}_1, \ldots, \hat{L}_{t_{\max}}]$ and obtain the denoised data $\hat{\mathcal{S}}_{\text{denoised}}$ and $\hat{\mathcal{L}}_{\text{denoised}}$. Based on applications, we output different results. For example, in the low-light denoising case, our output is $\hat{S}$ since the goal is to extract out the sparse targets. In traditional denoising scenarios, the output can be $\hat{\mathcal{L}}_{\text{denoised}}$ or $\hat{\mathcal{I}}_{\text{denoised}}$=$\hat{\mathcal{S}}_{\text{denoised}}$+$\hat{\mathcal{L}}_{\text{denoised}}$. This depends on whether the video contains only background or back-

Algorithm 5 ReProCS-based Layering Denoising (ReLD)

Splitting:

1. Initialization using PCP [10]: Compute $(\hat{\mathcal{L}}_0, \hat{\mathcal{S}}_0) \leftarrow \text{PCP}(\mathcal{M}_0)$ and compute $[\hat{P}_0, \hat{\Sigma}_0] \leftarrow \text{approx-basis}(\hat{\mathcal{L}}_0, 90\%)$. The notation $\text{PCP}(\mathcal{M})$ means implementing the PCP algorithm on matrix $\mathcal{M}$ and $P = \text{approx-basis}(\mathcal{M}, q\%)$ means that P is the $q\%$ left singular vectors' matrix for $\mathcal{M}$.

2. For all $t > t_0$, implement an appropriately modified ReProCS algorithm
   (a) Split M_t into layers $\hat{L}_t$ and $\hat{S}_t$:
      i. Compute $y_t \leftarrow \Phi_t M_t$ with $\Phi_t \leftarrow I - \hat{P}_{t-1}\hat{P}_{t-1}'$
      ii. Compute $\hat{S}_t$ as the solution of
      $$\min_x \|x\|_1 \text{s.t.} \|y_t - \Phi_t x\|_2 \leq \xi$$
      with $\xi = \|\Phi_t \hat{L}_{t-1}\|$
      iii. $\hat{\mathcal{T}}_t \leftarrow \text{Thresh}(\hat{S}_t, \omega)$ with $\omega = 3\sqrt{\|M_t\|^2/n}$. Here $\mathcal{T} \leftarrow \text{Thresh}(x, \omega)$ means that $\mathcal{T} = \{i : |(x)_i| \geq \omega\}$
      $\hat{S}_{t,*} \leftarrow \text{LS}(y_t, \Phi_t, \hat{\mathcal{T}}_t)$. Here $\hat{x} \leftarrow \text{LS}(y, A, \mathcal{T})$ means that $\hat{x}_{\mathcal{T}} = (A_{\mathcal{T}}' A_{\mathcal{T}})^{-1} A_{\mathcal{T}}' y$, which is least-squared estimate of x on $\mathcal{T}$.
      iv. $\hat{L}_t \leftarrow M_t - \hat{S}_t$, $\hat{L}_{t,*} \leftarrow M_t - \hat{S}_{t,*}$
   (b) Perform subspace update, i.e., update $\hat{P}_t$ using projection-PCA introduced in Chapter.4

VBM3D Denoising:

1. $\hat{\sigma}_{\text{fg}} \leftarrow \text{Std-est}([\hat{S}_t, \ldots, \hat{S}_{t_0}])$
   $\hat{\sigma}_{\text{bg}} \leftarrow \text{Std-est}([\hat{L}_t, \ldots, \hat{L}_{t_0}])$. Here $\text{Std-est}(\mathcal{M})$ denotes estimating the standard deviation of noise from $\mathcal{M}$: we first subtract column-wise mean from $\mathcal{M}$ and then compute the standard deviation by seeing it as a vector.

2. $\hat{\mathcal{S}}_{\text{denoised}} \leftarrow \text{VBM3D}([\hat{S}_1, \ldots, \hat{S}_{t_{\max}}], \hat{\sigma}_{\text{fg}})$
   $\hat{\mathcal{L}}_{\text{denoised}} \leftarrow \text{VBM3D}([\hat{L}_1, \ldots, \hat{L}_{t_{\max}}], \hat{\sigma}_{\text{bg}})$. Here $\text{VBM3D}(\mathcal{M}, \sigma)$ implements the VBM3D algorithm on matrix $\mathcal{M}$ with input standard deviation σ.

Output: $\hat{\mathcal{S}}$, $\hat{\mathcal{S}}_{\text{denoised}}$, $\hat{\mathcal{L}}_{\text{denoised}}$ or $\hat{\mathcal{I}}_{\text{denoised}} = \hat{\mathcal{S}}_{\text{denoised}} + \hat{\mathcal{L}}_{\text{denoised}}$ based on applications

ground and foreground. In practice, even for videos with only backgrounds, adding $\hat{\mathcal{S}}_{\text{denoised}}$ helps improve PSNR.

Subspace Update phase (Optional). In long videos the span of the L_t's will change with time. Hence one needs to update the subspace estimate $\hat{P}_t$ every so often. This can be done efficiently using the projection-PCA algorithm from Chapter.4.

6.4 Experiments

Video demos and all tables of peak signal to noise ratio (PSNR) comparisons are also available at http://www.ece.iastate.edu/~hanguo/denoise.html. Code for ReLD is also posted on the page. The same code with all the same parameters is used for all our experiments.

Table 6.1: Comparing PSNRs and running time in seconds for Waterfall video (small size). Format: PSNR using $\hat{\mathcal{I}}_{\text{denoised}}$, PSNR using $\hat{\mathcal{L}}_{\text{denoised}}$ and (running time in seconds). All results are for videos of dimension $108 \times 192 \times 650$. The running time for DnCNN and MLP does not include training time.

Gaussian	Dataset: waterfall										
	ReLD	PCP-LD	SPCP-LD	GRASTA-LD	RPCA-GD-LD	ORPCA	AltProj-LD	VBM3D	MLP	DnCNN	SALT
25	**35.00**, 32.78(72.13)	**34.92**, 32.84(198.49)	34.90, 32.99(291.59)	30.42, 28.29(57.62)	34.06, 32.25(80.97)	30.77, 7.04(678.44)	33.34, 31.98(99.54)	32.02(24.32)	28.26(487.37)	28.27(4616)	31.99(8492)
30	**34.51**, 32.68(72.27)	**34.42**, 32.60(182.22)	34.39, 32.74(261.96)	29.39, 27.30(56.90)	33.55, 32.12(79.91)	29.97, 7.00(687.05)	32.53, 31.53(104.41)	30.95(23.41)	29.96(486.33)	27.51(4509)	31.11(8507)
50	**33.07**, 32.27(72.72)	**32.93**, 31.65(195.54)	32.90, 31.79(277.97)	25.31, 22.77(56.84)	32.32, 31.64(80.99)	27.99, 6.86(730.05)	30.48, 30.09(126.91)	27.99(23.69)	18.87(487.44)	25.55(4462)	29.00(8533)
70	29.25, **31.79**(69.15)	29.17, 30.67(198.73)	29.29, 30.82(292.59)	22.50, 22.40(54.18)	28.86, **31.02**(77.94)	24.32, 6.75(763.08)	27.97, 29.63(131.27)	24.42(20.65)	15.03(486.25)	24.32(4470)	27.26(8526)
S&P	ReLD	PCP-LD	SPCP-LD	GRASTA-LD	RPCA-GD-LD	ORPCA	AltProj-LD	VBM3D	MLP	DnCNN	SALT
$20+5\%$	31.50, 32.32(70.44)	31.70, **33.03**(199.21)	31.65, **33.13**(408.03)	28.83, 29.00(57.53)	31.32, 32.33(80.62)	27.09, 7.03(728.71)	31.35, 32.65(97.13)	27.74(24.33)	21.94(486.02)	24.80(5143)	18.83(8125)
$20+10\%$	29.43, 31.94(72.34)	29.79, **32.94**(201.50)	29.86, **33.07**(473.63)	27.07, 28.15(57.53)	29.51, 32.21(81.19)	25.18, 6.98(743.11)	29.46, 32.45(118.84)	25.70(24.11)	18.10(487.57)	23.46(5329)	15.68(8076)
$20+20\%$	24.45, 30.64(70.98)	24.77, **32.74**(208.44)	24.83, **32.87**(536.65)	22.90, 25.92(55.86)	24.49, 29.36(79.42)	21.66, 6.90(746.62)	24.66, 32.31(133.82)	21.84(22.07)	14.41(494.54)	22.39(5480)	12.15(8236)

Table 6.2: Comparing PSNRs and running time in seconds for Waterfall video (mid size). Format: PSNR using $\hat{\mathcal{I}}_{\text{denoised}}$, PSNR using $\hat{\mathcal{L}}_{\text{denoised}}$ and (running time in seconds) All results are for videos of dimension $540 \times 960 \times 100$. Notice we had to use only 100 frames when the image size was increased because the batch Robust PCA solutions - PCP, SPCP, AltProj, RPCA-GD - run out of memory with the full 650 frame video. ReProCS-LD (ReLD) does not, but to keep comparisons uniform we used only 100 frames for all.

Gaussian	Dataset: waterfall										
	ReLD	PCP-LD	SPCP-LD	GRASTA-LD	RPCA-GD-LD	ORPCA	AltProj-LD	VBM3D	MLP	DnCNN	SALT
25	**33.84**, 29.98(317.97)	33.38, 29.17(409.99)	33.36, 29.22(1039.7)	29.49, 12.58(387.58)	31.40, 28.87(383.04)	28.99, 9.56(3692)	28.99, 27.54(485.29)	33.67(104.30)	31.11(1311)	31.26(15214)	**34.05**(37281)
30	**33.01**, 29.79(327.92)	32.49, 28.72(389.95)	32.47, 28.76(1136.1)	28.11, 11.65(395.59)	30.29, 28.42(385.91)	27.53, 9.59(3796)	27.63, 26.58(485.29)	32.75(105.69)	29.18(1337)	30.40(15205)	**33.07**(34313)
50	**30.49**, 28.86(325.66)	29.79, 26.83(378.07)	29.77, 26.82(1445.2)	22.18, 9.46(379.30)	27.43, 26.64(390.78)	22.31, 9.66(4111)	23.74, 23.29(526.19)	30.18(105.29)	19.00(1346)	28.28(14051)	**30.67**(34810)
70	27.39, **27.77**(312.41)	26.80, 25.06(370.31)	26.78,25.02(1662.3)	13.57, 9.06(372.93)	25.04, 25.25(378.93)	20.61, 9.65(4417)	20.95, 20.88(570.44)	26.75(96.29)	15.08(1347)	27.06(14108)	**28.70**(35771)

Table 6.3: Comparing PSNRs and running time in seconds for Waterfall video (original size). Format: PSNR using $\hat{\mathcal{I}}_{\text{denoised}}$, PSNR using $\hat{\mathcal{L}}_{\text{denoised}}$ and (running time in seconds). All results are for videos of dimension $1080 \times 1920 \times 100$. The running time for DnCNN and MLP does not include training time. We did not implement algorithms that have memory restriction problems here.

Gaussian	Dataset: waterfall					
	ReLD	PCP-LD	AltProj-LD	GRASTA-LD	VBM3D	MLP
25	**35.13**, 29.79 (2.83×10^3)	34.52, 29.03 (6.31×10^3)	30.69, 28.50 (2.88×10^3)	29.94, 10.72 (1.83×10^3)	**36.04** (533)	33.73 (5.56×10^3)
30	**34.14**, 29.61 (2.34×10^3)	33.45, 28.59 (5.50×10^3)	29.22, 27.62 (2.38×10^3)	28.46, 10.42 (1.88×10^3)	**35.18** (550)	30.94 (5.49×10^3)
50	**31.10**, 28.72 (2.31×10^3)	30.31, 26.75 (4.89×10^3)	25.18, 24.58 (2.45×10^3)	22.27, 8.69 (1.98×10^3)	**32.55** (536)	19.06 (5.50×10^3)
70	**27.71**, 27.66 (2.38×10^3)	27.08, 25.00 (4.82×10^3)	22.23, 22.09 (2.48×10^3)	13.59, 8.33 (1.98×10^3)	**28.45** (608)	15.09 (5.55×10^3)

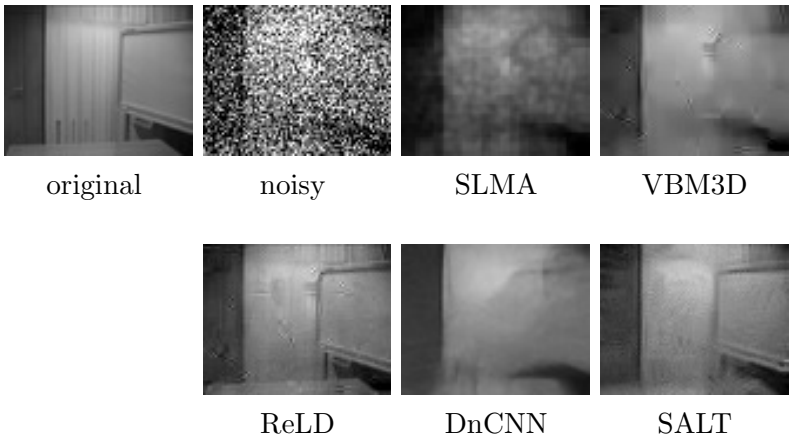
Table 6.4: PSNR (and running time in second) for different denoising algorithms on datasets of fountain, escalator, curtain and lobby.

σ	Dataset: fountain					Dataset: escalator				
	ReLD	VBM3D	DnCNN	SALT	SLMA	ReLD	VBM3D	DnCNN	SALT	SLMA
25	**32.67**(16.70)	**31.18**(5.44)	27.50(1081.3)	31.01(1956.5)	22.93(3.05×10^4)	**31.01**(16.64)	30.32(5.34)	26.40(1127.8)	**30.33**(1979.5)	21.17(3.09×10^4)
30	**32.25**(15.84)	**30.26**(5.17)	26.50(1049.8)	30.12(1960.1)	21.85(3.06×10^4)	**30.27**(16.45)	29.29(5.38)	25.31(1080.8)	**29.30**(1984.1)	20.49(3.15×10^4)
50	**30.53**(15.82)	26.55(5.24)	24.07(1029.0)	**27.69**(1964.1)	18.55(3.13×10^4)	**27.84**(16.03)	25.10(5.27)	22.45(1057.3)	**26.48**(2000.9)	17.98(3.21×10^4)
70	**27.53**(15.03)	22.08(4.69)	22.52(1031.5)	**25.92**(1962.9)	16.25(3.19×10^4)	**25.15**(15.28)	20.20(4.72)	20.53(1043.4)	**24.45**(2006.2)	15.90(3.18×10^4)
σ	Dataset: curtain					Dataset: lobby				
	ReLD	VBM3D	DnCNN	SALT	SLMA	ReLD	VBM3D	DnCNN	SALT	SLMA
25	**35.47**(16.78)	34.60(4.15)	31.77(1253.2)	32.27(1495.9)	23.28(7.75×10^4)	**39.78**(57.96)	**35.00**(19.57)	29.88(3482.4)	34.62(6489.2)	23.43(3.75×10^5)
30	**34.58**(17.35)	33.59(4.37)	30.55(1242.2)	34.27(1496.6)	22.74(9.05×10^4)	**38.76**(57.99)	**33.64**(19.09)	28.86(3403.1)	33.44(6474.3)	21.15(3.82×10^5)
50	**31.91**(17.17)	30.29(4.42)	28.03(1240.7)	31.60(1497.0)	19.12(7.86×10^4)	**35.15**(58.41)	29.23(19.35)	26.62(3354.1)	**30.47**(6568.8)	18.21(3.99×10^5)
70	28.10(16.50)	26.15(3.85)	26.42(1247.9)	**29.19**(1497.6)	16.68(8.30×10^4)	**29.68**(56.51)	24.90(17.00)	25.21(3365.6)	**28.16**(6581.8)	16.82(4.09×10^5)

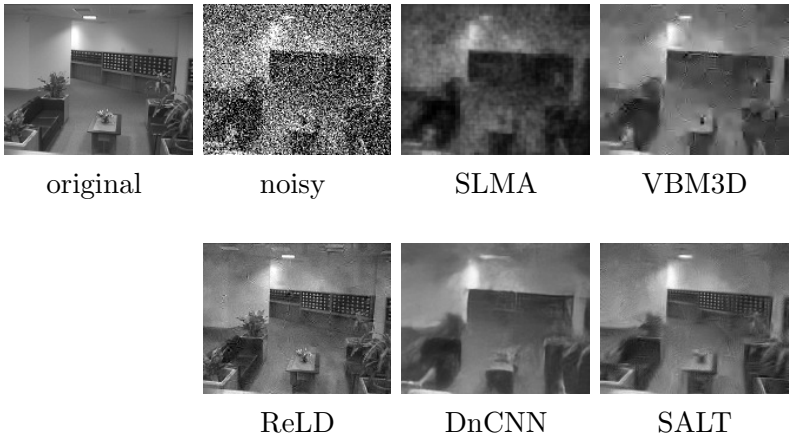
Remove Gaussian noise. First, with different levels of Gaussian noise, we compare performance of our proposed denoising framework with video layering performed using either ReProCS, or using the other robust PCA algorithms - PCP [10], SPCP [115], AltProj [66], RPCA-GD [108], ORPCA [25] and GRASTA [33]. We call the respective algorithms ReLD, PCP-LD, SPCP-LD, AltProj-LD, RPCA-GD-LD, ORPCA-LD and GRASTA-LD for short. We test all these on the waterfall dataset (downloaded from Youtube https://www.youtube.com/watch?v=UwSzu_0h7Bg). Besides these Layering-Denoising algorithms, we also compare with VBM3D [21], SALT [98] and two neural network image denoising methods, DnCNN [112] and MLP [5]. For MLP, we use masks that are trained from image patches that were corrupted with Gaussian noise with $\sigma = 25$ and hence the denoising performance is best with $\sigma = 25$ and deteriorates for other noise levels. For DnCNN we used masks for each noise level so the comparisons are fair. We did not re-train any neural network, so the times reported are only testing times. Notice both are much slower than ReLD.

The waterfall video is slowly changing and hence is well modeled as being low-rank. We add i.i.d. Gaussian noise with different variances to the video. The video consists of 650 frames and the original image size is 1080×1920. Due to memory restriction, some algorithms cannot be tested for this size. Hence for a full comparison we under-sampled the image and compared the algorithms with image size 108×192 and 540×960. We summarize the results for image size 108×192 with $\sigma = 25, 30, 50$, and 70 in Table 6.1. The full results with two other different image sizes (not all algorithms were compared) are provided in Table.6.2 and Table.6.3. For LD algorithms, we found that outputting $\hat{\mathcal{L}}_{\text{denoised}}$ as denoising results has higher PSNRs when noise variance is large. Therefore we display PSNRs computed using $\hat{\mathcal{L}}_{\text{denoised}}$ for $\sigma = 70$ for Layering-Denoising algorithms. As can been seen in Table 6.1 (first 4 rows), ReLD has either the best performance or close to the best one.

Gaussian noise on more datasets. From the experimental results shown above, ReLD was the best LD solution in terms of both speed and performance. Hence we only compare ReLD with the other non-LD approaches in this experiment. In Table 6.4, we provide comparisons on four



(a) Curtain



(b) Lobby

Figure 6.6: Visual comparison of denoising performance for Curtain and Lobby dataset for very large Gaussian noise ($\sigma = 70$)

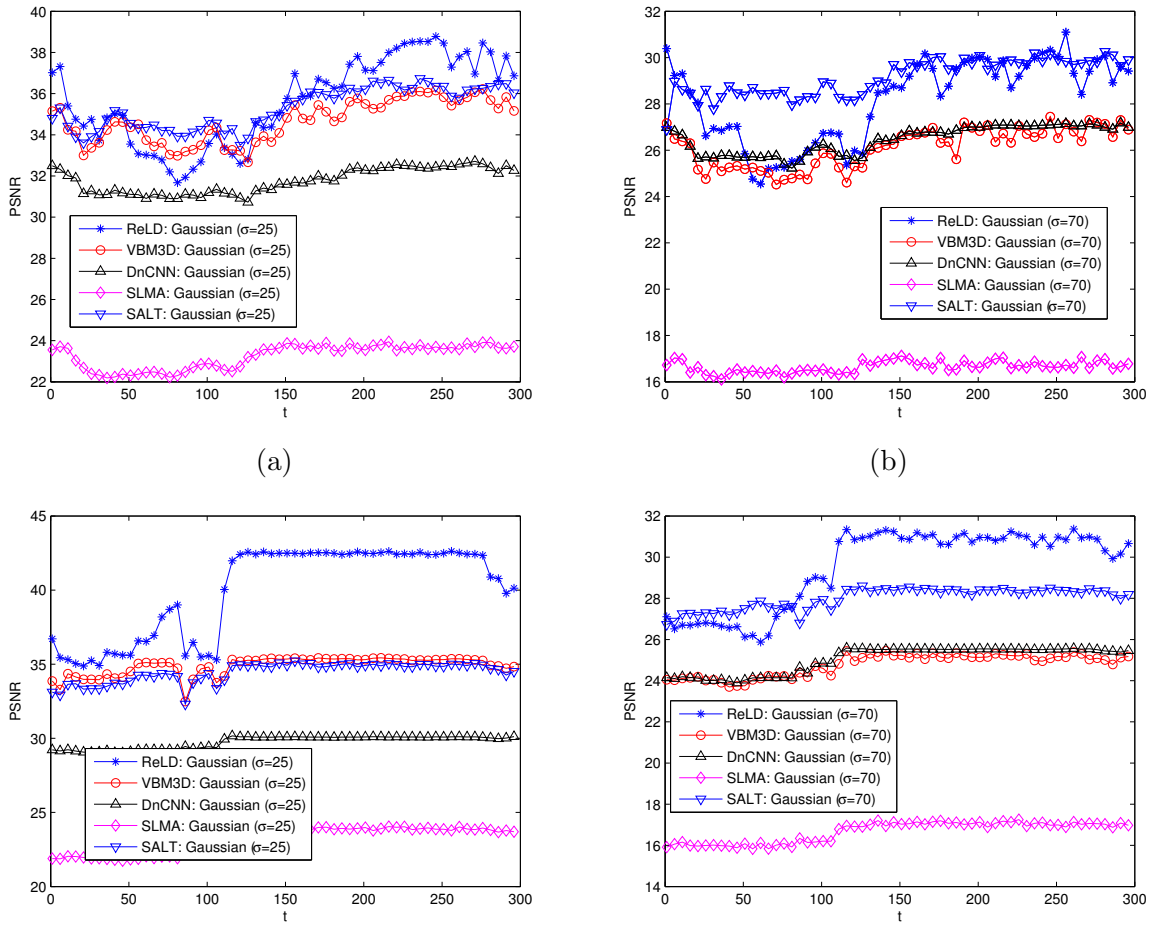


Figure 6.7: Frame-wise PSNR for Curtain and Lobby dataset with different noise level: (a) Curtain, Gaussian, $\sigma = 25$, (b) Curtain, Gaussian, $\sigma = 70$, (c) Lobby, Gaussian, $\sigma = 25$, (d) Lobby, Gaussian, $\sigma = 70$

more videos - fountain, escalator, curtain and lobby with different levels of Gaussian noise. In Fig. 6.7 we plot frame-wise PSNRs for the curtain and lobby datasets with diferent noise levels. In Fig.6.5a and Fig.6.5b we show sample visual comparisons for the curtain and lobby dataset which is also corrupted by Gaussian noise with $\sigma = 70$. As can be seen, ReLD and SALT are the top two algorithms with highest PSNRs and are able to recover more details of the images while other algorithms either fail or cause severe blurring. For curtain dataset with Gaussian ($\sigma = 70$) noise, as shown in Table 6.4, the PSNRs for ReLD and SALT are 28.10 and 29.19, respectively. In this case, SALT has slightly higher PSNR, but a careful visual comparison in Fig. 6.5a shows that ReLD recovers sharper details. The desk in the bottom of the image frame is recovered more clearly by ReLD than by SALT. In terms of running time, ReLD is significantly faster too.

Removing Salt & **Pepper noise.** We compare denoising performance on videos corrupted by salt-and-pepper noise. We compare the same algorithms as in previous experiments, and also use the same waterfall dataset. We corrupted it with 5%, 10% and 20% salt and pepper noise and then added Gaussian noise to it with $\sigma = 20$. The result is also summarized in Table 6.1. Here we also display PSNRs computed using $\hat{\mathcal{L}}_{\text{denoised}}$ for the Layering-Denoising algorithms.

low-light environment. In this part we test the ReLD's performance for denoising in low-light environment, i.e., to see target signal in the low-light environment. The video was taken in a dark environment where a barely visible person walked through the scene. The output we are using here for ReLD is $\hat{\mathcal{S}}$, which is the output of ReProCS. In Fig.6.8 we see ReLD is able to enhance the visual quality – observing the walking person. Note that Histogram-Equalization in this case can generate extra noise and cannot automatically mark the target.

6.5 Conclusion

From our experiments, (i) ReLD is most powerful for videos containing small-sized images, because, these are the videos for which VBM3D has a hard time finding enough matching patches to average over. (ii) It is also most useful for denoising of very noisy videos, e.g., Gaussian noise standard deviation equal to 70 (for 0-255 pixel value range), or of videos corrupted by salt-and-

Figure 6.8: Ability of "seeing" in the dark for two sample frames. From left to right: orignal dark image, results by ReLD, and Histogram-Equalization.

pepper noise. In both these cases, finding correctly matching patches is hard. Averaging over wrong matches of course results in bad denoised video quality. In conclusion, for small-sized or very noisy videos, ReLD should be used, while for very large sized images or small noise, just VBM3D is a better idea except if the layers are also needed. ReLD has the extra advantage of also outputting estimates of the background and foreground layers.

CHAPTER 7. A FUTURE WORK DIRECTION

The idea of "low-rank + sparse" is widely applied in various applications e.g., video foreground/background separation, fMRI imaging, voice separation and anomaly detection. It's popular due to its interpretability – many problems in nature have such a "low-rank + sparse" property.

In machine learning terminology, "low-rank + sparse" solutions, or robust PCA solutions are unsupervised and do not need the process of training using well-labelled data, which can be an advantage if there is no access to annotated data or the data size is too small. However, in many tasks such as image classification and object detection, large-scaled well-annotated data is publicly accessble, compared with deep learning methods, robust PCA solutions are usually not preferred due to not utilizing the large amount of data.

To address this limit, there has been works studying the combination of robust PCA and deep learning. One interesting example is in [93] where the authors show that, the performance of video object detection task can be improved if the deep learning detector is performed on the foreground mask, which is first separated out by an robust PCA method. Specifically, the authors used an PCP based robust PCA method to perform the foreground/background separation and next implemented faster-RCNN on the foreground mask to detect the object. The faster-RCNN model was first trained with a large-scaled dataset. The robust PCA step in this example can be seen as a pre-processing step, and the idea is very similar to that of layering-denoising introduced in Chapter 6. In future research, it will be interesting to see how prac-ReProCS can be combined with deep learning approaches in solving industrial problems, either as a pre-processing step or as a key component.

In the following part of this chapter, we discuss the problem of logo detection, a computer vision task that is independent of the discussion of prac-ReProCS in previous chapters. However, in developping the current solution to the logo detection problem, we found that a combination of

prac-ReProCS and faster-RCNN, a state-of-the-art object detection algorithm, can be a valuable solution which makes the current one from semi-automatic to fully-automatic. The work was developped under internship projects at Adobe Research in 2018 and 2019 and is under submission to a double-blind reviewed conference on multimedia.

7.1 Introduction

Logo detection is an important problem in a wide range of applications, e.g., vehicle logo recognition for intelligent traffic-control systems [103, 67, 80, 81], consumer perception [88], copyright protection and infringement detection [40, 110, 19]. The majority of the logo detection solutions are closed-set approaches, where all different brands of logos are known and are pre-trained within the detection system. On the contrary, a category of solutions are designed for open-set detection tasks where logos are not seen and predefined in the system [91, 95]. The main idea of the open-set solutions is to consider all the input logo instances as one class (with the other class being images without any logos), and train a simple binary logo detector. Being able to handle a large amount of diverse unseen logos and brands is the biggest advantage of the open-set approaches. In real-world applications, however, open-set solutions are not widely used due to their high false-positive rates – common non-logo objects that share similar shapes or textures with logos (e.g., a round clock, a red cross) are very likely to be detected as logos.

As a special case of general object detection task, logo detection methods are becoming more accurate and robust with recent development of object detection algorithms. Deep neural-network architectures enable detecting objects and logos even in diverse contexts such as presence of different illuminations and occlusions. However, proper training of deep neural-network based algorithms require large amount of training data per class with object-level annotations, which prevents many existing algorithms from considering a larger number of logo classes (see Fig.7.1). For this reason, they are not scalable to real-world or industry-level logo detection tasks where a much larger number of logo classes are of interest.

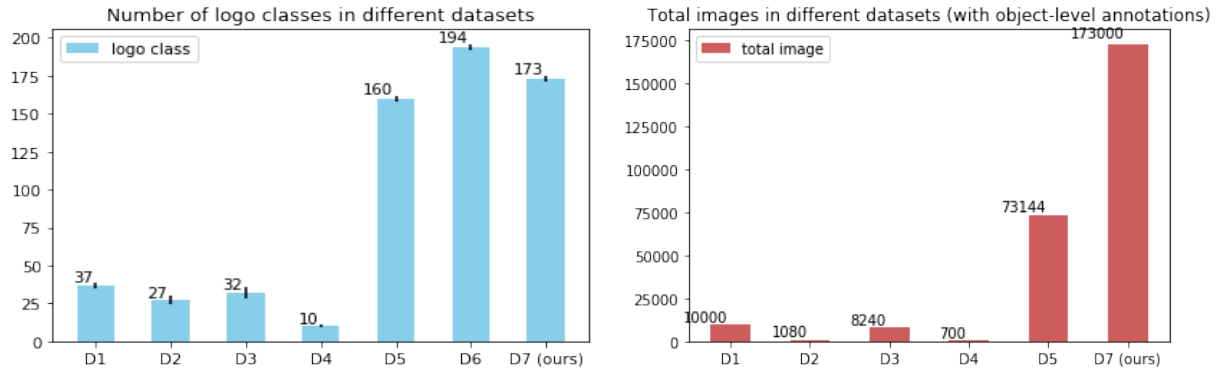


Figure 7.1: D1: BelgaLogos[46], D2: FlickrLogos-27 [47], D3: FlickrLogos-32 [81], D4: TopLogo-10 [90], D5: LOGO-NET [35], D6: WebLogo-2M [89], **D7: Logo173 (ours).** Left: comparison of number of logo classes in different datasets. Right: comparison of total images in different datasets (with object-level annotation). D6 has a total of 1,867,177 images but they are all labelled in image level and contain noise. Therefore we do not plot it in the right chart.

In a recent work [89], to avoid exhaustive manual labelling, a novel incremental learning approach called Scalable Logo Self-training (SLST) was proposed. The algorithm explores the webly data (images obtained from image search queries in Twitter) and learns by iteratively self-mining training images from noisy web data and updating current model. Applying this learning principle, it introduced a very large logo dataset called "WebLogo-2M". Although it requires no annotation work, there are still obstacles that hinder it from being widely applied to industry-level applications. Firstly, SLST framework runs the self-mining and model-update iterations with the assumption that the number of logo classes is fixed while in reality it is more practical to desire that new logo classes can be added to the system. Secondly, SLST, as a completely automatic system, inevitably has the model drift problem - the errors in model prediction can be propagated through the iterations. Thirdly, the dataset that the authors introduced is noisy and not labelled at the object level and hence has limited value for a quick implementation.

To tackle these problems, in this work we consider a distributed data augmentation and training pipeline where each logo class is trained and updated independently before finally training a multi-class logo detector. For each logo class, we initially bootstrap using synthetic data. This is followed by iteratively self-mining (selecting good samples) from weakly-labelled image candidates

Figure 7.2: Illustration of logo detection challenges

and updating current model. A multi-class detector is trained when each logo class finishes its iterations and collects enough real image data. Treating each logo class independently brings two benefits: (1) The training and updating task for each logo class can be implemented using different hardware, which achieves computation efficiency and makes it possible to add new logo classes without affecting existing tasks. (2) If human curation is included to avoid model drift, curating one logo class at a time is advantageous due to ease of visual display (discussed later).

Our contribution in this work are three-fold: (1) We introduce a semi-automatic solution to the logo detection problem which finds a balance between reducing annotation work and avoiding model drift. To our best knowledge, this is the first work that use semi-automatic training strategy in logo detection tasks. (2) We propose a pipeline for data augmentation and training, which initially treats each logo class independently and thus achieves computation efficiency and allows new logo classes to be added to the system later. This architecture also makes human curation easier - we develop a simple User Interface where pre-selected results are zoomed to the bounding-box level and displayed as an image array, and users can conveniently pick incorrect samples by simple clicks. (3) With this pipeline, we also created a large dataset Logo173 that includes 173,000 images of 173 logo classes (1000 images per logo), and all images are well-annotated with bounding-box

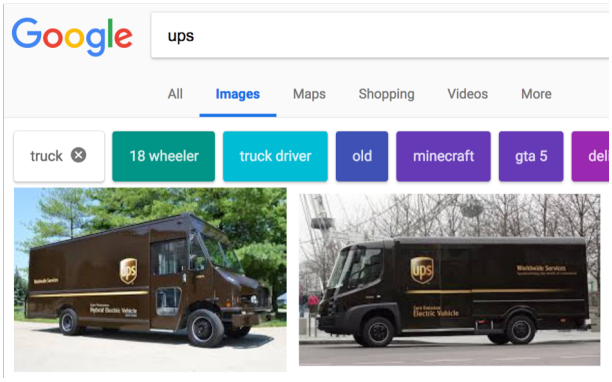


Figure 7.3: Example of obtaining weakly-labelled data from Google Image Search.

information. In terms of the number of logo classes and total images, this is the largest dataset that has object-level bounding-box information.

The idea of using **weakly-labelled** data can be seen in many applications [17, 50, 89, 86, 44, 84, 106, 83]. Images obtained from search engines are usually pre-filtered by search queries and thus contain relevant information. For instance, images obtained from Google Image Search by the query "UPS" are mostly related to the shipping company UPS (see Fig.7.3). The term "weakly-labelled" comes from the fact that neither do such images have object-level bounding box information nor are they guaranteed to even have the corresponding logos. Although weakly-labelled, they are great candidates to make a well-annotated training set.

"Headshot logos" vs "logos in the scene". In a practical setting, the logo detection task involves detecting logos in natural scenes which are very often complex. This entails that the images we collect for training to be representative across various kinds of natural scenes. One can clearly see the difference between a "headshot logo" image and "logo in the scene" image as shown in Fig.7.5. To avoid getting two many "headshot logos" images, we carefully designed meaningful search queries instead of merely searching with logo names.

Specifically, for each logo, we generate a set of search queries which takes into consideration factors such as their search frequency in history and popularity among media coverage. For instance, for the "Apple" logo, a set of meaningful search queries may include "Apple store", "Apple event 2019", "Apple new product 2019", etc.

Figure 7.4: 173 logo classes we selected in the Logo173 dataset.

Figure 7.5: Two different kinds of logo images. Left: "headshot logo"; Right: "logo in natural scene".

Logo Class. A total of 173 logo classes are selected in the Logo173 dataset (Fig.7.4). We choose these logos largely based on their popularity. One thing to note is that our training pipeline allows new logos to be added to the system conveniently which makes expanding the number of logo classes easy.

Image Source. Weakly-labelled images can come from search engines such as Google and Bing, or from social media networks e.g., Facebook, Instagram and Twitter. There is a clear trade-off between image scene complexity and logo instance richness. In general, images containing logos in social media have a more natural and complex scene. Such images are captured and uploaded by individual users (including amateur and professional photographers) and tend to cover a variety of scenes. But some logo instances are difficult to see in social media if they are not popular in daily topics of users.

Among all social media, we found that Twitter is the only platform that provides an API to download images in batches. Unlike Su et al [89], we did not choose Twitter as our image source for the following two reasons. Firstly, we found that images on twitter are more likely to be noisy, which means the images obtained by logo search queries are less likely to contain relevant logos. It would require us to download a huge amount of image data to get enough logo instances. Secondly, as discussed earlier, some logo classes can be very sparse and we are not able to obtain even the minimum number of images required for training a certain logo class, and hence we need to deal with unbalanced data. Therefore we use Google Image Search to collect weakly labelled data, and by simple scripts we were able to scrape adequate candidate images using meaningful search queries mentioned above. This was followed by a de-duplication step where we removed duplicated images by computing their simple feature vectors which contain basic statistics such as image size, mean pixel values in three channels and pixel values at a set of predefined positions.

7.2 semi-automatic data augmentation and training pipeline

In this work, our goal is to train a multi-class logo detection model when initially only weakly-labelled data is available. The core step is to efficiently convert the weakly-labelled data into

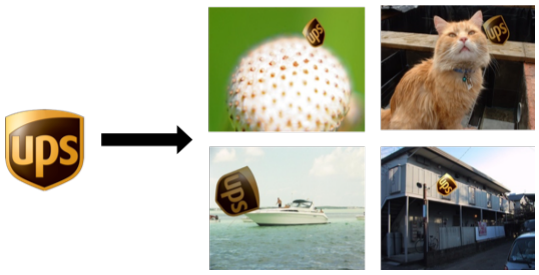


Figure 7.6: Illustration of generating synthetic data. The "UPS" icon is transformed in different ways and pasted to various kinds of background images.

well-labelled (object-wise) data such that it can be trained using state-of-the-art object-detection methods. In a nutshell, our pipeline is designed to firstly generate well-labelled training data and secondly train a logo detector with the obtained data.

The training data is obtained independently from each single logo class through an iterative process where we incrementally enhance the model capability for each logo. This is achieved by training an initial model using synthetic data, followed by iteratively self-mining (from weak-labelled data) and updating the current model.

Specifically, in our implementation for logo detection, we choose Faster-RCNN [78] for its excellent performance for ordinary object detection tasks (if the object is not too small) [114]. One can also use other alternatives such as SSD [56] and YOLO [77]. Next, we introduce the pipeline as following and it is summarized in detail in Algorithm 6.

7.2.1 Initial training with synthetic data

Using synthetic data for deep-neural-network training has been widely applied for many applications. In all the tasks, synthetic data is generated in a way such that it is representative and that the gap (in terms of image style, object style) between synthetic data and real data is minimized.

Apparently, synthetic data generated for logo detection has a smaller object style gap than that for general object detection. This is due to the simple nature that logos has smaller in-class variations compared to other objects such as animals. It is usually common and valid to assume that most logos are in rigid 2D shapes and have less deformation variations like animal images (see

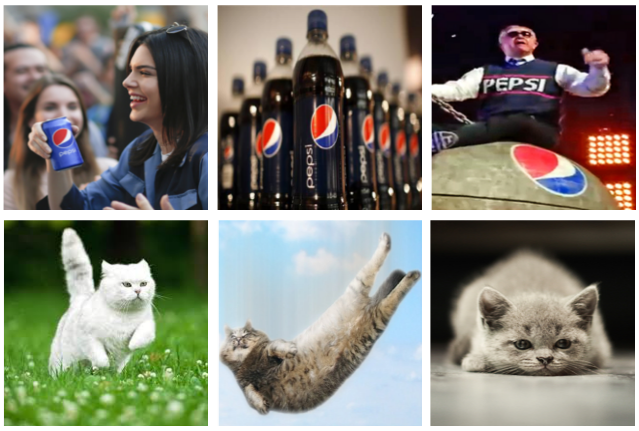


Figure 7.7: Comparison of logos and cats in images. First row: "pepsi" logos in three different images; Second row: cats in three different images. Logos are less variant than animals.

Fig.7.7). For this reason, we generate our synthetic training data by simple superimposing of logos on images.

Particularly, we generate synthetic training data by overlaying a standard logo image onto randomly selected background images after various kinds of image transformation as illustrated in Fig.7.6. Note that there are other methods to generate synthetic data [68, 29, 32]. Since our goal is to eventually collect a large number of real image data which will replace the synthetic data, we believe a simple superimposing should suffice for model initialization. We leave the full comparison of all synthetic methods to a future follow-up work.

For each logo class, we generated 200 synthetic images with the bounding box information. A binary logo detector for each class was trained using Faster-RCNN with the synthetic data.

7.2.2 Iteratively self-mining and updating the current model

With each initial logo detector, we run inference on the weakly-labelled images and selected high-score results which are larger than a pre-defined threshold (we used 0.9) and marked them as new training data. The process is shown in Fig.7.8.

The newly-obtained training data is added to the training set to replace an equal number of synthetic images. The model is re-trained using the updated training data. We iteratively perform

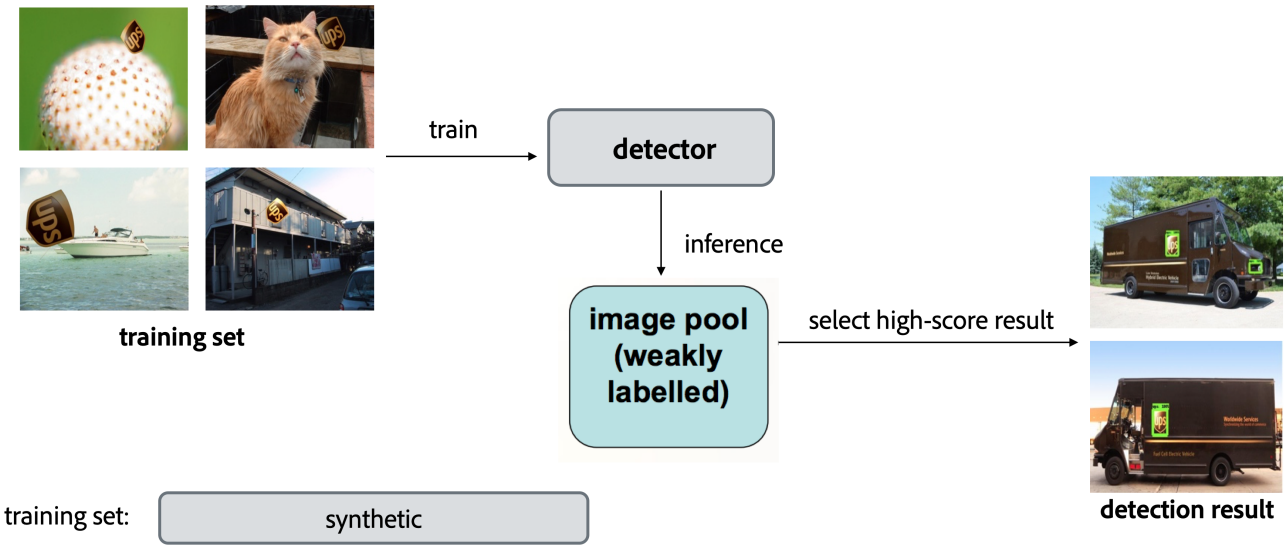


Figure 7.8: Illustration of training using synthetic data. We ran inference on the weakly-labelled images and selected high-score results which are larger than a pre-defined threshold and mark them as new training data.

self-mining (discovering new training data) and updating the current model until the training set for each logo class consists of only real images. We keep doing this until each training set contains more than 1000 real images. These real images with object-level bounding box information are used for training the final multi-class logo detector.

7.2.3 Curation (optional): removing false-positives

The key to the data augmentation and training pipeline is accurately self-selecting trustworthy images from the weakly-labelled data at each iteration. Although at each step the inference results with highest detection scores are picked, they are not guaranteed to be all correct. This may cause the so-called model drift effect – the errors in the model prediction is propagated through the iterations and eventually introduces bias to the multi-class logo detector.

The errors that are accumulated through all steps are mainly caused by false-positives in the logo detection case. In the pipeline, the false-positives are treated as correct training samples for retraining the model. The most effective way to avoid this from happening is to manually curate the

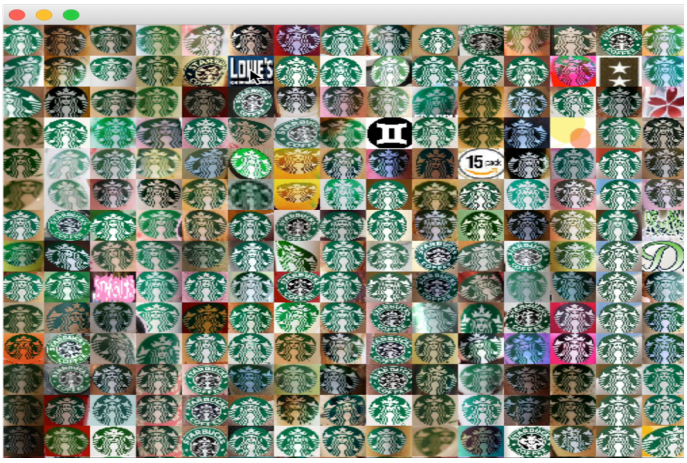


Figure 7.9: User Interface for curation. We display the high-score detection results for "Starbucks" in an image array, where each patch corresponds to a bounding-box in the detection results. The outliers are removed by simple-clicks.

Figure 7.10: Example of false negative. The four bounding-boxes are successful detections while there is a false-negative at top-left corner.

result and filter out the false-positives. Human curation may sound daunting, but it is effortless in this case since it simply requires to pick a few outliers from a large group of correct data. As shown in Fig.7.9, we display the high-score detection results for "Starbucks" in an image array, where each patch corresponds to a bounding-box in the detection results. It can been seen that selecting non-Starbucks patches from the array is an easy task - in the User-Interface that we designed, they are deleted by simple clicks.

Another detection error that is not easily noticeable is a false negative – the logos that the detector fails to detect. In Fig.7.10, the detector is able to detect four logos but misses the top-left one and these four bounding-boxes are sent to training set for model re-training and updating. The potential problem in this case is that the undetected logo at the top-left corner – without a logo label, can be marked as a negative sample by the Region Proposal Network in Faster-RCNN. We assume such false negatives rarely occur in a single image and hence ignore their effects.

7.2.4 Training the multi-class logo detector

Once collecting the real training data is finished for each logo class, we start training the multi-class logo detector using a standard Faster-RCNN model.

7.3 Experiments

We compared our semi-automatic data augmentation and training pipeline with a similar scheme SLST [89]. Both pipelines were implemented with Faster-RCNN as the core detection algorithm. In this work, we aim to propose an efficient data augmentation and training pipeline instead of developing a new object detection algorithm. Therefore, we do not aim to compare the performance of different object detection algorithms for logo data. We only implemented Faster-RCNN for this part.

Algorithm 6 Training Pipeline for Logo Detection

Input: n standard logo icons, weakly-labelled images for each logo class, background images for generating synthetic training data

Initialization: For each logo class i, generate 200 synthetic images by randomly superimposing the logo icon to any background images. Let T_i denote the training set for logo class i, and $T_i = S_i \cup R_i$ where S_i and R_i are set of synthetic and real image data, respectively. $|S_i| = 200$.

Step 1: For each logo class i:

Initial Training: Train a model $\mathcal{F}_i^{(0)}$ using data from S_i

while $|T_i| < 1000$ **do**

1. Inference using $\mathcal{F}_i^{(iter)}$ on weakly-labelled dataset W_i: $W_{top}^{(iter)} = \mathcal{F}_i^{(iter)}(W_i)$;
2. Update the training set:

   $R_i \leftarrow R_i \cup W_{top}^{(iter)}$;

   $W_i \leftarrow W_i \setminus W_{top}^{(iter)}$;

   Keep removing same number of images from S_i until it is empty
3. Re-train model with new data:

   $\mathcal{F}_i^{(iter+1)} \leftarrow \mathcal{F}_i^{(iter)}$

Here $W_{top}^{(iter)} = \mathcal{F}_i^{(iter)}(W_i)$ means selecting the top detection results using detector $\mathcal{F}_i^{(iter)}$ and we use a threshold of 0.9 for detection scores; $R_i \cup W_{top}^{(iter)}$ and $W_i \setminus W_{top}^{(iter)}$ are set union and set minus operations to add new data to the training set and remove images from the weakly-labelled image pool.

Step 2: Training multi-class logo detector

With data $T_1, T2, \cdots, T_n$ (all consist of real images), train a multi-class logo detector using faster-RCNN model

Output: A multi-class logo detector

Figure 7.11: A single receptive field such as that of the RPN cannot match the object scale variability. This is also an example where our detector was not able to detect the logo due to small size but was later succeeded after doing up-sampling.

Table 7.1: Performance comparison of different pipelines

Model	mAP
SLST	0.69
our pipeline	0.74
our pipeline with curation	0.81

Table 7.2: Class-wise mAP (%) in different iterations. Iter0 refers to the initial model trained using synthetic data. Values in parentheses denote mAPs obtained with the curation step.

	starbucks	ups	mcdonalds	kfc	pepsi	bmw	benz	walmart	google
iter0	25.44	23.11	22.31	20.13	19.32	20.15	23.13	24.11	18.22
iter1	32.11 (38.1)	29.87 (40.31)	30.76 (41.22)	28.54 (35.22)	27.43 (38.41)	29.18 (37.23)	30.33 (35.81)	34.76 (42.39)	23.89 (33.99)
iter2	45.39 (60.77)	48.92 (59.87)	51.77 (66.21)	44.39 (58.73)	43.98 (62.38)	37.58 (58.21)	42.31 (57.77)	40.69 (60.13)	38.44 (54.34)

Performance Metrics We use mean Average Precision (mAP) to evaluate the detection performance. By convention we apply the 0.5 IoU rule - a detection is considered correct if the Intersection over Union (IoU) between the detection and groundtrue exceeds 50%.

Implementation details For initial training with synthetic data, we generated 200 training images per class. In all training tasks using Faster-RCNN, we set the learning rate to be 0.0001. We

Table 7.3: mAPs on different testing groups based on logo scales. We split our testing set based on logo size scales. Specifically we computed the logo-to-image size ratio for each image and divided the images into three different groups: $0-0.2$ (small), $0.2-0.4$ (medium) and $0.4-0.8$ (large). We summarize different mAPs for each image group in column 2. In column 3, we summarize the results after applying the enhancement technique based on up-sampling.

Logo size ratio	mAP (curation)	mAP (curation + enhancement)
0 - 0.2	0.32	0.68
0.2 - 0.4	0.85	0.87
0.4 -0.8	0.77	0.77

used the ResNet-101 architecture [34] and the network was pre-trained on the COCO dataset [52]. The comparison was on a testing set that contains 3460 images and were independently labelled.

We summarized the detection performance in TABLE.7.3. Particularly, we compared two versions of our pipeline - with / without the curation step to remove the false-positives. It can be clearly seen that adding the curation step improves logo detection performance since model drift can be prevented to some extent. In Table.7.2, we tracked the class-wise mAPs along iterations and the data shows that the multiple iterations help improve the model capacity.

Note that our baseline outperforms SLST and we believe one of the reasons is the difficulty in transforming the multi-class logo detection model trained from synthetic data to real-world images. Intuitively, with pure synthetic data only, in order to well adapt the initial model to real-world images, one would prefer a group of binary-class detectors such that the gap between real data and synthetic data can be absorbed into each class instead of aggregating in a cumbersome multi-class detector. Moreover, along the iterations, there are more cross-class false-positives when using a multi-class detector, and such error can be accumulated and cause more severe model drift.

Another notable phenomenon we observed is the unsatisfactory detection performance on small logos, which has been recognized as a major drawback of faster-RCNN. The RPN in faster-RCNN generates region proposals by sliding a fixed set of filters over a fixed set of convolutional feature maps which creates an inconsistency between different sizes of objects (logos)[6]. The logo sizes are variable while the filter receptive fields are fixed. As shown in Fig.7.11, a fixed receptive field is not able to cover the multiple scales where logos appear in natural scenes. To understand the detection

performance on logos in different size scales, we split our testing set based on logo size scales and compute the mAPs for each group respectively and the results are summarized in Table.7.3 (column 2). As can be seen in the table, the performance was pegged back due to small logo sizes.

A simple method to handle this is by up-sampling the input image both during training and testing [6]. In our loops for data-collection and re-training, we tried cropping (zooming) small-sized logo images and only took local regions into training. This also avoids the logo feature distortion introduced by image reshape in Faster-RCNN. In detection, we tried pre-segmenting the input image into 4×4 sub-images (with overlaps) and ran detection on the 4+1 images. These two extra steps were able to compensate the degradation caused by small logo size as show in Table.7.3 (column 3). There are other recent works focused on small-sized object detection problems [85, 37, 20]. Since our goal in this work is to show the efficiency of the pipeline that incorporates data-augmentation and training, it is out of our scope to compare all detection algorithms.

7.4 Conclusion

We present a scalable data augmentation and training pipeline for logo detection. The pipeline explores the weakly-labelled data from image search engines and is able to save the effort for exhaustive manual labelling. With the human curation step, our pipeline is only semi-automatic, but it considers both reducing annotation work and avoiding model drift. Moreover, we construct a large logo detection benchmarking dataset Logo173 with the above mentioned semi-automatic manner.

In our future work, we desire to replace the human curation step with other unsupervised machine learning techniques to make it a fully-automatic pipeline. In Figure 7.9 we show the API used for picking the outliers. In deed, picking the outliers from the image-patch matrix can be considered as a robust-PCA problem, after each patch is aligned properly [69]. An interesting idea is to apply prac-ReProCS to it and kick out the outliers as a sparse signal, such that the data-augmentation and training pipeline is fully-automatic.

BIBLIOGRAPHY

[1] (2014). Blinded title. In *Double blind conference submission.*

[2] Agostinelli, F., Anderson, M., and Lee, H. (2013). Adaptive multi-column deep neural networks with application to robust image denoising. In *Advances in Neural Information Processing Systems*, pages 1493–1501.

[3] Brand, M. (2002). Incremental singular value decomposition of uncertain data with missing values. In *European Conference on Computer Vision*, pages 707–720.

[4] Buades, A., Coll, B., and Morel, J. (2005). Image denoising by non-local averaging. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP'05). IEEE International Conference on*, volume 2, pages 25–28. IEEE.

[5] Burger, H., Schuler, C., and Harmeling, S. (2012). Image denoising: Can plain neural networks compete with bm3d? In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2392–2399. IEEE.

[6] Cai, Z., Fan, Q., Feris, R. S., and Vasconcelos, N. (2016). A unified multi-scale deep convolutional neural network for fast object detection. In *European conference on computer vision*, pages 354–370. Springer.

[7] Candes, E. (2008). The restricted isometry property and its implications for compressed sensing. *Compte Rendus de l'Academie des Sciences, Paris, Serie I*, pages 589–592.

[8] Candes, E. and Tao, T. (2005). Decoding by linear programming. *IEEE Trans. Info. Th.*, 51(12):4203 – 4215.

[9] Candes, E. and Tao, T. (2007). The dantzig selector: statistical estimation when p is much larger than n. *Annals of Statistics*, 35 (6):2313–2351.

[10] Candès, E. J., Li, X., Ma, Y., and Wright, J. (2011). Robust principal component analysis? *Journal of ACM*, 58(3).

[11] Candès, E. J. and Recht, B. (2012). Exact matrix completion via convex optimization. *Commun. ACM*, 55(6):111–119.

[12] Chandrasekaran, V., Recht, B., Parrilo, P. A., and Willsky, A. S. (2012). The convex geometry of linear inverse problems. *Foundations of Computational Mathematics*, (6).

[13] Chandrasekaran, V., Sanghavi, S., Parrilo, P. A., and Willsky, A. S. (2011). Rank-sparsity incoherence for matrix decomposition. *SIAM Journal on Optimization*, 21.

[14] Chatterjee, P. and Milanfar, P. (2009). Clustering-based denoising with locally learned dictionaries. *Image Processing, IEEE Transactions on*, 18(7):1438–1451.

[15] Chatterjee, P. and Milanfar, P. (2010). Is denoising dead? *IEEE Transactions on Image Processing*, 19(4):895–911.

[16] Chen, S., Donoho, D., and Saunders, M. (1998). Atomic decomposition by basis pursuit. *SIAM Journal of Scientific Computing*, 20:33–61.

[17] Chen, X. and Gupta, A. (2015). Webly supervised learning of convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1431–1439.

[18] Chen, X., Song, L., and Yang, X. (2016). Deep rnns for video denoising. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 9971.

[19] Choi, Y.-H. and Choi, T.-S. (2005). Robust logo embedding technique for copyright protection. In *Consumer Electronics, 2005. ICCE. 2005 Digest of Technical Papers. International Conference on*, pages 341–342. IEEE.

[20] Cui, L. (2018). Mdssd: Multi-scale deconvolutional single shot detector for small objects. *arXiv preprint arXiv:1805.07009*.

[21] Dabov, K., Foi, A., and Egiazarian, K. (2007a). Video denoising by sparse 3d transform-domain collaborative filtering. In *European Signal Processing Conference*, volume 149. Tampere, Finland.

[22] Dabov, K., Foi, A., Katkovnik, V., and Egiazarian, K. (2007b). Image denoising by sparse 3-d transform-domain collaborative filtering. *Image Processing, IEEE Transactions on*, 16(8):2080–2095.

[23] Elad, M. and Aharon, M. (2006). Image denoising via sparse and redundant representations over learned dictionaries. *Image Processing, IEEE Transactions on*, 15(12):3736–3745.

[24] Feng, J., Xu, H., Mannor, S., and Yan, S. (2013a). Online pca for contaminated data. In *Adv. Neural Info. Proc. Sys. (NIPS)*.

[25] Feng, J., Xu, H., and Yan, S. (2013b). Online robust pca via stochastic optimization. In *Adv. Neural Info. Proc. Sys. (NIPS)*.

[26] Foi, A., Katkovnik, V., and Egiazarian, K. (2007). Pointwise shape-adaptive dct for high-quality denoising and deblocking of grayscale and color images. *Image Processing, IEEE Transactions on*, 16(5):1395–1411.

[27] Friedlander, M., Mansour, H., Saab, R., and Yilmaz, O. (2012). Recovering compressively sampled signals using partial support information. *IEEE Trans. Info. Th.*, 58(2):1122–1134.

[28] Ganesh, A., Min, K., Wright, J., and Ma, Y. (2012). Principal component pursuit with reduced linear measurements. In *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, pages 1281–1285. IEEE.

[29] Georgakis, G., Mousavian, A., Berg, A. C., and Kosecka, J. (2017). Synthesizing training data for object detection in indoor scenes. *arXiv preprint arXiv:1702.07836*.

[30] Guo, H., Qiu, C., and Vaswani, N. (2014). An online algorithm for separating sparse and low-dimensional signal sequences from their sum. *IEEE Trans. Sig. Proc.*, 62(16):4284–4297.

[31] Guo, L., Au, O., Ma, M., and Liang, Z. (2007). Temporal video denoising based on multihypothesis motion compensation. *Circuits and Systems for Video Technology, IEEE Transactions on*, 17(10):1423–1429.

[32] Gupta, A., Vedaldi, A., and Zisserman, A. (2016). Synthetic data for text localisation in natural images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2315–2324.

[33] He, J., Balzano, L., and Szlam, A. (2012). Incremental gradient on the grassmannian for online foreground and background separation in subsampled video. In *IEEE Conf. on Comp. Vis. Pat. Rec. (CVPR)*.

[34] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[35] Hoi, S. C., Wu, X., Liu, H., Wu, Y., Wang, H., Xue, H., and Wu, Q. (2015). Logo-net: Large-scale deep logo detection and brand recognition with deep region-based convolutional networks. *arXiv preprint arXiv:1511.02462*.

[36] Hsu, D., Kakade, S. M., and Zhang, T. (2011). Robust matrix decomposition with sparse corruptions. *Information Theory, IEEE Transactions on*, 57(11):7221–7234.

[37] Hu, G. X., Yang, Z., Hu, L., Huang, L., and Han, J. M. (2018). Small object detection with multiscale features. *International Journal of Digital Multimedia Broadcasting*, 2018.

[38] Hu, Y., Goud, S., and Jacob, M. (2012). A fast majorize-minimize algorithm for the recovery of sparse and low-rank matrices. *IEEE Transactions on Image Processing*, 21(2):742=753.

[39] I.T., J. (2002). *Principal Component Analysis*. Springer, second edition.

[40] Jabade, V. S. and Gengaje, S. R. (2012). Logo based image copyright protection using discrete wavelet transform and fuzzy inference system. *International Journal of Computer Applications*, 58(10).

[41] Ji, H., Huang, S., Shen, Z., and Xu, Y. (2011). Robust video restoration by joint sparse and low rank matrix approximation. *SIAM Journal on Imaging Sciences*, 4(4):1122–1142.

[42] Ji, H., Liu, C., Shen, Z., and Xu, Y. (2010). Robust video denoising using low rank matrix completion. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1791–1798. IEEE.

[43] Jia, K., Chan, T.-H., and Ma, Y. (2012). Robust and practical face recognition via structures sparisty. In *Eur. Conf. on Comp. Vis. (ECCV)*.

[44] Jia, S., Lansdall-Welfare, T., and Cristianini, N. (2016). Gender classification by deep learning on millions of weakly labelled images. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 462–467. IEEE.

[45] Joachimiak, M., Rusanovskyy, D., Hannuksela, M., and Gabbouj, M. (2012). Multiview 3d video denoising in sliding 3d dct domain. In *Signal Processing Conference (EUSIPCO), 2012 Proceedings of the 20th European*, pages 1109–1113. IEEE.

[46] Joly, A. and Buisson, O. (2009). Logo retrieval with a contrario visual query expansion. In *Proceedings of the 17th ACM international conference on Multimedia*, pages 581–584. ACM.

[47] Kalantidis, Y., Pueyo, L. G., Trevisiol, M., van Zwol, R., and Avrithis, Y. (2011). Scalable triangulation-based logo recognition. In *Proceedings of the 1st ACM International Conference on Multimedia Retrieval*, page 20. ACM.

[48] Khajehnejad, A., Xu, W., Avestimehr, A., and Hassibi, B. (2009). Weighted ℓ_1 minimization for sparse recovery with prior information. In *IEEE Intl. Symp. Info. Th. (ISIT)*.

[49] Lee, K. and Bresler, Y. (2010). Admira: Atomic decomposition for minimum rank approximation. *IEEE Transactions on Information Theory*, 56(9).

[50] Li, D., Huang, J.-B., Li, Y., Wang, S., and Yang, M.-H. (2016). Weakly supervised object localization with progressive domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3512–3520.

[51] Li, Y., Xu, L., Morphett, J., and Jacobs, R. (2003). An integrated algorithm of incremental and robust pca. In *IEEE Intl. Conf. Image Proc. (ICIP)*, pages 245–248.

[52] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer.

[53] Lin, Z., Chen, M., and Ma, Y. (November 2009a). Alternating direction algorithms for l1 problems in compressive sensing. Technical report, University of Illinois at Urbana-Champaign.

[54] Lin, Z., Ganesh, A., Wright, J., Wu, L., Chen, M., and Ma, Y. (August 2009b). Fast convex optimization algorithms for exact recovery of a corrupted low-rank matrix. Technical report, University of Illinois at Urbana-Champaign.

[55] Liu, C. and Freeman, W. (2010). A high-quality video denoising algorithm based on reliable motion estimation. In *European Conference on Computer Visioin*, pages 706–719. Springer.

[56] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer.

[57] Lois, B., Vaswani, N., and Qiu, C. (2013). Performance guarantees for undersampled recursive sparse recovery in large but structured noise. In *GlobalSIP*.

[58] Mairal, J., Bach, F., Ponce, J., Sapiro, G., and Zisserman, A. (2009). Non-local sparse models for image restoration. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2272–2279. IEEE.

[59] Mardani, M., Mateos, G., and Giannakis, G. (2013a). Dynamic anomalography: Tracking network anomalies via sparsity and low rank. *J. Sel. Topics in Sig. Proc.*

[60] Mardani, M., Mateos, G., and Giannakis, G. B. (2013b). Recovery of low-rank plus compressed sparse matrices with application to unveiling traffic anomalies. *IEEE Trans. Info. Th.*

[61] Mateos, G. and Giannakis, G. (2012). Robust pca as bilinear decomposition with outlier-sparsity regularization. *IEEE Trans. Sig. Proc.*

[62] McCoy, M. and Tropp, J. A. (2011). Two proposals for robust pca using semidefinite programming. *Electronic Journal of Statistics*, 5:1123–1160.

[63] McCoy, M. B. and Tropp, J. A. (2012). Sharp recovery bounds for convex deconvolution, with applications. *arXiv:1205.1580, accepted to J. Found. Comput. Math.*

[64] Narayanamurthy, P. and Vaswani, N. (to appear, 2018). Provable dynamic robust pca or robust subspace tracking. *IEEE Trans. Info. Theory.*

[65] Needell, D. and Tropp., J. (2009). Cosamp: Iterative signal recovery from incomplete and inaccurate samples. *Appl. Comp. Harmonic Anal.*, 26(3):301–321.

[66] Netrapalli, P., Niranjan, U. N., Sanghavi, S., Anandkumar, A., and Jain, P. (2014). Non-convex robust pca. In *NIPS*.

[67] Pan, C., Yan, Z., Xu, X., Sun, M., Shao, J., and Wu, D. (2013). Vehicle logo recognition based on deep learning architecture in video surveillance for intelligent traffic system.

[68] Peng, X., Sun, B., Ali, K., and Saenko, K. (2015). Learning deep object detectors from 3d models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1278–1286.

[69] Peng, Y., Ganesh, A., Wright, J., Xu, W., and Ma, Y. (2012). Rasl: Robust alignment by sparse and low-rank decomposition for linearly correlated images. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2233–2246.

[70] Poor, H. V. (1994). *An Introduction to Signal Detection and Estimation.* Springer, second edition.

[71] Qiu, C. and Vaswani, N. (2010). Real-time robust principal components' pursuit. In *Allerton Conf. on Communications, Control and Computing.*

[72] Qiu, C. and Vaswani, N. (2011). Recursive sparse recovery in large but correlated noise. In *Allerton Conf. on Communication, Control, and Computing.*

[73] Qiu, C. and Vaswani, N. (2013). Recursive sparse recovery in large but structured noise – part 2. In *IEEE Intl. Symp. Info. Th. (ISIT)*.

[74] Qiu, C., Vaswani, N., Lois, B., and Hogben, L. Recursive robust pca or recursive sparse recovery in large but structured noise. *under revision for IEEE Trans. Info. Th., also at arXiv: 1211.3754[cs.IT], shorter versions in ICASSP 2013 and ISIT 2013*.

[75] Rabbani, H. and Gazor, S. (2012). Video denoising in three-dimensional complex wavelet domain using a doubly stochastic modelling. *IET image processing*, 6(9):1262–1274.

[76] Rahman, S., Ahmad, M., and Swamy, M. (2007). Video denoising based on inter-frame statistical modeling of wavelet coefficients. *Circuits and Systems for Video Technology, IEEE Transactions on*, 17(2):187–198.

[77] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.

[78] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99.

[79] Richard, E., Savalle, P.-A., and Vayatis, N. Estimation of simultaneously sparse and low rank matrices. *arXiv:1206.6474, appears in Proceedings of the 29th International Conference on Machine Learning (ICML 2012)*.

[80] Romberg, S. and Lienhart, R. (2013). Bundle min-hashing for logo recognition. In *Proceedings of the 3rd ACM conference on International conference on multimedia retrieval*, pages 113–120. ACM.

[81] Romberg, S., Pueyo, L. G., Lienhart, R., and Van Zwol, R. (2011). Scalable logo recognition in real-world images. In *Proceedings of the 1st ACM International Conference on Multimedia Retrieval*, page 25. ACM.

[82] Roweis, S. (1998). Em algorithms for pca and spca. *Advances in Neural Information Processing Systems*, pages 626–632.

[83] Sapienza, M., Cuzzolin, F., and Torr, P. H. (2014). Learning discriminative space–time action parts from weakly labelled videos. *International journal of computer vision*, 110(1):30–47.

[84] Shi, Z., Hospedales, T. M., and Xiang, T. (2013). Bayesian joint topic modelling for weakly supervised object localisation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2984–2991.

[85] Singh, B., Najibi, M., and Davis, L. S. (2018). Sniper: Efficient multi-scale training. In *Advances in Neural Information Processing Systems*, pages 9310–9320.

[86] Siva, P., Russell, C., and Xiang, T. (2012). In defence of negative mining for annotating weakly labelled data. In *European Conference on Computer Vision*, pages 594–608. Springer.

[87] Skocaj, D. and Leonardis, A. (2003). Weighted and robust incremental method for subspace learning. In *IEEE Intl. Conf. on Computer Vision (ICCV)*, volume 2, pages 1494 –1501.

[88] Soomro, Y. A. and Shakoor, R. (2011). Impact of logo on consumer perception of a company. *Interdisciplinary Journal of Contemporary Research In Business*, 3(7):61–81.

[89] Su, H., Gong, S., Zhu, X., et al. (2018a). Weblogo-2m: Scalable logo detection by deep learning from the web.

[90] Su, H., Zhu, X., and Gong, S. (2017). Deep learning logo detection with data expansion by synthesising context. In *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*, pages 530–539. IEEE.

[91] Su, H., Zhu, X., and Gong, S. (2018b). Open logo detection challenge. *arXiv preprint arXiv:1807.01964*.

[92] Tao, M. and Yuan, X. (2011). Recovering low-rank and sparse components of matrices from incomplete and noisy observations. *SIAM Journal on Optimization*, 21(1):57–81.

[93] Tejada, E. D. and Rodriguez, P. A. (2017). Moving object detection in videos using principal component pursuit and convolutional neural networks. In *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 793–797. IEEE.

[94] Torre, F. D. L. and Black, M. J. (2003). A framework for robust subspace learning. *International Journal of Computer Vision*, 54:117–142.

[95] Tüzkö, A., Herrmann, C., Manger, D., and Beyerer, J. (2017). Open set logo detection and retrieval. *arXiv preprint arXiv:1710.10891*.

[96] Vaswani, N. and Lu, W. (2010). Modified-cs: Modifying compressive sensing for problems with partially known support. *IEEE Trans. Signal Processing*.

[97] Waters, A. E., Sankaranarayanan, A. C., and Baraniuk, R. G. (2011). Sparcs: Recovering low-rank and sparse matrices from compressive measurements. In *Proc. of Neural Information Processing Systems(NIPS)*.

[98] Wen, B., Li, Y., Pfister, L., and Bresler, Y. (2017). Joint adaptive sparsity and low-rankness on the fly: An online tensor reconstruction scheme for video denoising. In *The IEEE International Conference on Computer Vision (ICCV)*.

[99] Wen, B., Ravishankar, S., and Bresler, Y. (2015). Video denoising by online 3d sparsifying transform learning. In *Image Processing (ICIP), 2015 IEEE International Conference on*, pages 118–122. IEEE.

[100] Wright, J., Ganesh, A., Min, K., and Ma, Y. (2013). Compressive principal component pursuit. *Information and Inference*, 2(1):32–68.

[101] Wright, J. and Ma, Y. (2010a). Dense error correction via l1-minimization. *IEEE Trans. on Info. Th.*, 56(7):3540–3560.

[102] Wright, J. and Ma, Y. (2010b). Dense error correction via l1-minimization. *IEEE Trans. Info. Th.*, 56(7):3540–3560.

[103] Xia, Y., Feng, J., and Zhang, B. (2016). Vehicle logo recognition and attributes prediction by multi-task learning with cnn. In *Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), 2016 12th International Conference on*, pages 668–672. IEEE.

[104] Xie, J., Xu, L., and Chen, E. (2012). Image denoising and inpainting with deep neural networks. In *Advances in Neural Information Processing Systems*, pages 341–349.

[105] Xu, H., Caramanis, C., and Sanghavi, S. (2012). Robust pca via outlier pursuit. *IEEE Tran. on Information Theory*, 58(5).

[106] Xu, Y., Kong, Q., Wang, W., and Plumbley, M. D. (2018). Large-scale weakly supervised audio classification using gated convolutional neural network. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 121–125. IEEE.

[107] Yang, J. and Zhang, Y. (June 2010). Alternating direction algorithms for l1 problems in compressive sensing. Technical report, Rice University.

[108] Yi, X., Park, D., Chen, Y., and Caramanis, C. (2016). Fast algorithms for robust pca via gradient descent. In *Advances in neural information processing systems*, pages 4152–4160.

[109] Yu, S., Ahmad, M., and Swamy, M. (2010). Video denoising using motion compensated 3-d wavelet transform with integrated recursive temporal filtering. *Circuits and Systems for Video Technology, IEEE Transactions on*, 20(6):780–791.

[110] Yuan, Y., Huang, D., and Liu, D. (2006). An integer wavelet based multiple logo-watermarking scheme. In *Computer and Computational Sciences, 2006. IMSCCS'06. First International Multi-Symposiums on*, volume 2, pages 175–179. IEEE.

[111] Zhan, J. and Vaswani, N. (2013). Separating sparse and low-dimensional signal sequence from time-varying undersampled projections of their sums. In *IEEE Intl. Conf. Acoustics, Speech, Sig. Proc. (ICASSP)*.

[112] Zhang, K., Zuo, W., Chen, Y., Meng, D., and Zhang, L. (2017). Beyond a Gaussian denoiser: Residual learning of deep CNN for image denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155.

[113] Zhang, T. and Lerman, G. (2013). A novel m-estimator for robust pca. *arXiv:1112.4863v3, to appear in Journal of Machine Learning Research*.

[114] Zhao, Z.-Q., Zheng, P., Xu, S.-t., and Wu, X. (2019). Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*.

[115] Zhou, Z., Li, X., Wright, J., Candes, E., and Ma, Y. (2010). Stable principal component pursuit. In *Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on*, pages 1518–1522. IEEE.

APPENDIX. DETAILED DISCUSSION OF WHY REPROCS WORKS

Define the subspace estimation error as

$$\mathrm{SE}(P, \hat{P}) := \|(I - \hat{P}\hat{P}')P\|_2$$

where both P and $\hat{P}$ are *basis matrices.* Notice that this quantity is always between zero and one. It is equal to zero when range($\hat{P}$) contains range(P) and it is equal to one if $\hat{P}'P_i = 0$ for at least one column of P.

Recall that for $t \in [t_j, t_{j+1} - 1]$, $P_t = [P_{(j-1)}R_j \setminus P_{(j),\mathrm{old}}, P_{(j),\mathrm{new}}]$. Thus, L_t can be rewritten as

$$L_t = P_{(j-1)}a_{t,*} + P_{(j),\mathrm{new}}a_{t,\mathrm{new}}$$

where $a_{t,\mathrm{new}} := P'_{(j),\mathrm{new}}L_t$ and $a_{t,*} := P'_{(j-1)}L_t$.

Let $c := c_{\max}$ and $r_j := r_0 + jc$. We explain here the key idea of why ReProCS works [74, 1]. Assume the following hold besides the assumptions of chapter 3.

1. Subspace change is detected immediately, i.e. $\hat{t}_j = t_j$ and $c_{j,\mathrm{new}}$ is known.
2. Pick a $\zeta \ll 1$. Assume that $\|L_t\|_2 \le \gamma_*$ for a γ_* that satisfies $\gamma_* \le 1/\sqrt{r_J\zeta}$. Since ζ is very small, γ_* can be very large.
3. Assume that $(t_{j+1} - t_j) \ge K\alpha$ for a K as defined below.
4. Assume the following model on the gradual increase of $a_{t,\mathrm{new}}$: for $t \in [t_j + (k-1)\alpha, t_j + k\alpha - 1]$, $\|a_{t,\mathrm{new}}\|_2 \le v^{k-1}\gamma_{\mathrm{new}}$ for a $1 < v \le 1.2$ and $\gamma_{\mathrm{new}} \ll \gamma_*$.
5. Assume that projection PCA "works" i.e. its estimates satisfy $\mathrm{SE}(P_{(j),\mathrm{new}}, \hat{P}_{(j),\mathrm{new},k-1}) \le 0.6^{k-1} + 0.4c\zeta$. The proof of this statement is long and complicated and is given in [74, 1].
6. Assume that projection PCA is done K times with K chosen so that $0.6^{K-1} + 0.4c\zeta \le c\zeta$.

Assume that at $t = t_j - 1$, $\mathrm{SE}(P_{(j-1)}, \hat{P}_{(j-1)}) \leq r_{j-1}\zeta \ll 1$. We will argue below that $\mathrm{SE}(P_{(j)}, \hat{P}_{(j)}) \leq r_j\zeta$. Since $r_j \leq r_0 + Jc$ is small, this error is always small and bounded.

First consider a $t \in [t_j, t_j + \alpha)$. At this time, $\hat{P}_t = \hat{P}_{(j-1)}$. Thus,

$$\begin{aligned}\|\beta_t\|_2 &= \|(I - \hat{P}_{t-1}\hat{P}_{t-1}')L_t\|_2 \\ &\leq \mathrm{SE}(P_{(j-1)}, \hat{P}_{(j-1)})\|a_{t,*}\|_2 + \|a_{t,\mathrm{new}}\|_2 \\ &\leq (r_{j-1}\zeta)\gamma_* + \gamma_{\mathrm{new}} \\ &\leq \sqrt{\zeta} + \gamma_{\mathrm{new}} \end{aligned} \tag{.1}$$

By construction, $\sqrt{\zeta}$ is very small and hence the second term in the bound is the dominant one. By the slow subspace assumption $\gamma_{\mathrm{new}} \ll \|S_t\|_2$. Recall that β_t is the "noise" seen by the sparse recovery step. The above shows that this noise is small compared to $\|S_t\|_2$. Moreover, using Lemma 3.2.2 and simple arguments [see [74, Lemma 6.6]], it can be shown that

$$\delta_s(\Phi_t) \leq \kappa_*^2 + r_{j-1}\zeta$$

is small. These two facts along with any RIP-based result for ℓ_1 minimization, e.g. [7], ensure that S_t is recovered accurately in this step. If the smallest nonzero entry of S_t is large enough, it is possible get a support threshold ω that ensures exact support recovery. Then, the LS step gives a very accurate final estimate of S_t and it allows us to get an exact expression for $e_t := S_t - \hat{S}_t$. Since $\hat{L}_t = M_t - \hat{S}_t$, this means that L_t is also recovered accurately and $e_t = \hat{L}_t - L_t$. This is then used to argue that p-PCA at $t = t_j + \alpha - 1$ "works".

Next consider $t \in [t_j + (k-1)\alpha, t_j + k\alpha - 1]$. At this time, $\hat{P}_t = [\hat{P}_{(j-1)}, \hat{P}_{(j),\mathrm{new},k-1}]$. Then, it is easy to see that

$$\begin{aligned}\|\beta_t\|_2 &= \|(I - \hat{P}_{t-1}\hat{P}_{t-1}')L_t\|_2 \\ &\leq \mathrm{SE}(P_{(j-1)}, \hat{P}_{(j-1)})\|a_{t,*}\|_2 + \mathrm{SE}(P_{(j),\mathrm{new}}, \hat{P}_{(j),\mathrm{new},k-1})\|a_{t,\mathrm{new}}\|_2 \\ &\leq (r_{j-1}\zeta)\ \gamma_* + (0.6^{k-1} + 0.4c\zeta)\ v^{k-1}\gamma_{\mathrm{new}} \\ &\leq \sqrt{\zeta} + 0.72^{k-1}\gamma_{\mathrm{new}} \end{aligned} \tag{.2}$$

Ignoring the first term, in this interval, $\|\beta_t\|_2 \leq 0.72^{k-1}\gamma_{\text{new}}$, i.e. the noise seen by the sparse recovery step decreases exponentially with every p-PCA step. This, along with a bound on $\delta_s(\Phi_t)$ (this bound needs a more complicated argument than that for $k = 1$, see [74, Lemma 6.6]), ensures that the recovery error of S_t, and hence also of $L_t = M_t - S_t$, decreases roughly exponentially with k. This is then used to argue that the p-PCA error also decays roughly exponentially with k.

Finally for $t \in [t_j + K\alpha, t_{j+1} - 1]$, because of the choice of K, we have that $\text{SE}(P_{(j),\text{new}}, \hat{P}_{(j),\text{new},K}) \leq c\zeta$. At this time, we set $\hat{P}_{(j)} = [\hat{P}_{(j)-1}, \hat{P}_{(j),\text{new},K}]$. Thus, $\text{SE}(P_{(j)}, \hat{P}_{(j)}) \leq \text{SE}(P_{(j-1)}, \hat{P}_{(j)-1}) + \text{SE}(P_{(j),\text{new}}, \hat{P}_{(j),\text{new},K}) \leq r_{j-1}\zeta + c\zeta = r_j\zeta$.