UC Riverside

UC Riverside Previously Published Works

Title

Generative Models for Low-Dimensional Video Representation and Reconstruction.

Permalink

https://escholarship.org/uc/item/3qn459c1

Authors

Hyder, Rakib

Asif, M Salman

Publication Date

2020

DOI

10.1109/TSP.2020.2977256

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Generative Models for Low-Dimensional Video Representation and Reconstruction

Rakib Hyder and M. Salman Asif

***Abstract*—Generative models have received considerable attention in signal processing and compressive sensing for their ability to generate high-dimensional natural image using low-dimensional codes. In the context of compressive sensing, if the unknown image belongs to the range of a pretrained generative network, then we can recover the image by estimating the underlying compact latent code from the available measurements. In practice, however, a given pretrained generator can only reliably generate images that are similar to the training data. To overcome this challenge, a number of methods have been proposed recently to use untrained generator structure as prior while solving the signal recovery problem. In this paper, we propose a similar method for jointly updating the weights of the generator and latent codes while recovering a video sequence from compressive measurements. We use a single generator to generate the entire video. To exploit the temporal redundancy in a video sequence, we use a low-rank constraint on the latent codes that imposes a low-dimensional manifold model on the generated video sequence. We evaluate the performance of our proposed methods on different video compressive sensing problems under different settings and compared them against some state-of-the-art methods. Our results demonstrate that our proposed methods provide better or comparable accuracy and low computational and memory complexity compared to the existing methods.**

***Index Terms*—Compressive sensing, generative model, video reconstruction, manifold embedding.**

I. Introduction

DEEP generative networks, such as autoencoders, generative adversarial networks (GANs), and variational autoencoders (VAEs), are now commonly used in almost every machine learning and computer vision task [**?**], [**?**], [**?**], [**?**]. One key idea in these generative networks is that they can learn to transform a low-dimensional feature vector (or latent code) into realistic images and videos. The *range* of the generated images is expected to be close to the true underlying distribution of training images. Once these networks are properly trained (which remains a nontrivial task), they can generate remarkable images in the trained categories of natural scenes.

In this paper, we propose to use a deep generative model for compact representation and reconstruction of videos from a small number of linear measurements. We assume that a generative network structure is available, which we represent as

$$x = G_\gamma(z) \equiv g_{\gamma_L} \circ g_{\gamma_{L-1}} \circ \cdots \circ g_{\gamma_1}(z). \tag{1}$$

R. Hyder and M. Asif are with the Department of Electrical and Computer Engineering, University of California, Riverside, CA, 92521 USA (e-mail: rhyde001@ucr.edu; sasif@ece.ucr.edu).

This work was supported in part by ONR under grant N00014-19-1-2264.
Manuscript received September 4, 2019; accepted February 20, 2020.

$G_\gamma(z)$ denotes the overall function for the deep network with L layers that maps a low-dimensional (latent) code $z \in \mathbb{R}^k$ into an image $x \in \mathbb{R}^n$ and $\gamma = \{\gamma_1, \ldots, \gamma_L\}$ represents all the trainable parameters of the deep network. $G_\gamma(\cdot)$ as given in (1) can be viewed as a cascade of L functions g_{γ_l} for $l = 1, \ldots, L$, each of which represents a mapping between input and output of the respective layer. An illustration of such a generator with $L = 5$ is shown in Figure 1.

We consider a general problem of recovering a video sequence from its linear measurements. Suppose we are given a sequence of measurements for $t = 1, \ldots, T$ as

$$y_t = A_t x_t + e_t, \tag{2}$$

where x_t denotes the t^{th} frame in the unknown video sequence, y_t denotes its observed measurements, A_t denotes the respective measurement operator, and e_t denotes noise or error in the measurements. Our goal is to recover the video sequence (x_t) from the available measurements (y_t). The recovery problem becomes especially challenging as the number of measurements (in y_t) becomes very small compared to the number of unknowns (in x_t). To ensure quality reconstruction in such settings, we need a compact (low-dimensional) representation of the unknown signal. Thus, we use the given generative model to represent the video sequence as $x_t = G_\gamma(z_t)$ and seek to recover the unknown sequence x_t by optimizing over x_t, z_t, and γ.

We demonstrate that even if we do not have a pretrained generative network, we can still reconstruct it by jointly optimizing over network weights γ and the latent codes z_t. We observe that when we optimize over latent code alongside network weights, the temporal similarity in the video frames is reflected in the latent code representation. To exploit similarities among the frames in a video sequence, we also include low-rank constraints on the latent codes. An illustration of different representations we use in this paper are shown in Figure 2.

A. Related Work

Compressive sensing refers to a broad class of problems in which we aim to recover a signal from a small number of measurements [**?**], [**?**], [**?**]. The canonical compressive sensing problem in (2) is inherently underdetermined, and we need to use some prior knowledge about the signal structure. Classical signal priors exploit sparse and low-rank structures in images and videos for their reconstruction [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**]. However, the natural images exhibits far richer nonlinear structures than sparsity alone. We focus on a newly emerging family of data-driven representation methods

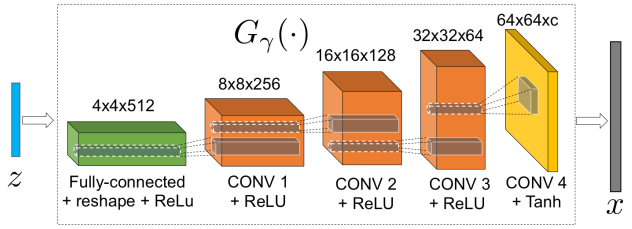


Fig. 1: A candidate architecture we use in our experiments with one fully connected and four fractionally strided convolutional layers. Generative model: $x = G_\gamma(z)$ maps a vector $z \in \mathbb{R}^k$ into an image $x \in \mathbb{R}^n$.

based on generative models that are learned from massive amounts of training data [?], [?], [?], [?], [?], [?].

Deep generative models have been extensively used for learning good representations for images and videos. Generative adversarial networks (GANs) and variational autoencoders (VAEs) are two popular classes of deep generative networks that learn a function that maps vectors drawn from a certain distribution in a low-dimensional space into images in a high-dimensional space [?], [?], [?], [?]. An attractive feature of autoencoders [?], [?], [?] and GANs [?], [?], [?], [?], [?], [?], [?], [?] is their ability to transform feature vectors to generate a variety of images from a different set of desired distributions. A number of generative models have been proposed to learn latent representation of an image with respect to a generator [?], [?], [?]. The learning process usually involves solving a nonlinear problem using gradient decent over latent codes to find the best approximation of the given image [?], [?].

The generative model and optimization problems we use are inspired by recent work on using generative models for compressive sensing in [?], [?], [?], [?], [?], [?], [?]. Compressive sensing using generative models was first introduced in [?], which used a trained deep generative network as a prior for image reconstruction from compressive measurements; the reconstruction problem involves optimization over the latent code of the generator. Since the generator is fixed, this approach works well only if the unknown image/video belongs to the range of the generator used. Deep image prior (DIP) is a related method in which an untrained convolutional generative model is used as a prior for solving inverse problems such as inpainting and denoising because of their tendency to generate natural images [?]; the reconstruction problem involves optimization of generator network parameters. Inspired by these observations, a number of methods have been proposed for solving compressive sensing problem by optimizing generator network weights while keeping the latent code fixed at a random value [?], [?]. Both DIP [?] and deep decoder [?] update the network parameters to generate a given image; therefore, the generator can reconstruct wide range of images. One key difference between the two approaches is that the network used in DIP is highly overparameterized, while the one used in deep decoder is underparameterized.

We observe two main limitations in the DIP and deep decoder-based video recovery that we seek to address in this paper. (1) The latent codes in DIP and deep decoder methods are initialized at random and stay fixed throughout the recovery process. Therefore, we cannot infer the structural similarities in the images from the structural similarities in the latent codes.

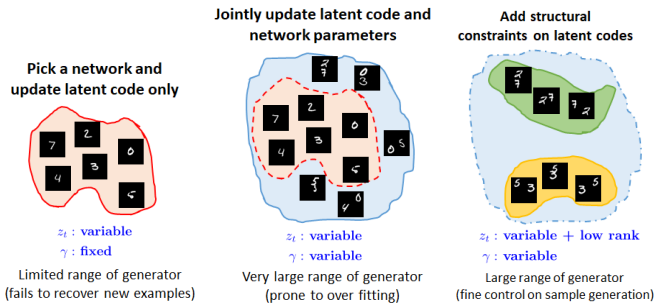


Fig. 2: An illustration of different generative priors discussed in the paper: (a) Optimizing latent codes can only reconstruct images in the range of the generative network. (b) Jointly optimizing latent code and network weights enables recovery of a larger range of images. (c) Low-rank and similarity constraints on latent code further regularize the problem and potentially explain other structures in data.

(2) Both of these methods train one network per image. A naïve approach to train one network per frame in a video will be computationally prohibitive, and if we train a single network to generate the entire video sequence, then their performance degrades.

B. Main Contributions

In this paper, we propose joint optimization of latent codes and generator weights along with low-rank constraint on the latent codes to solve video compressive sensing problems. We use a generative model, as described in (1), to find compact representation of videos in the form of z_t. To reconstruct a video sequence from the compressive measurements in (2), we jointly optimize over the latent codes z_t and the network parameters γ. Since the frames in a video sequence exhibit rich redundancies in their representation, we impose a low-rank constraint on the latent codes to represent the video sequence with a more compact representation of the latent codes.

The key contributions of this paper are as follows.

- Latent code optimization can only reconstruct a video sequence that belong to its range. We demonstrate that by jointly optimizing the latent codes with the network weights, we can expand the range of the generator and reconstruct images that the given initial generator fails on. We show that even though the network has a very large number of parameters, the joint optimization still converges to a good solution.
- Consecutive frames in a video sequence share lot of similarities. To encode similarities among the reconstructed frames, we introduce low-rank constraints on the generator latent codes. This enables us to represent a video sequence with a very small number of parameters in the latent codes and reconstruct them from a very small number of measurements.

II. Technical Approach

Let us assume that $x_t \in \mathbb{R}^n$ for $t = 1, \ldots, T$ is a sequence of video frames that we want to reconstruct from the measurements $y_t = A_t x_t + e_t$ as given in (2). The generative model as given in (1) maps a low-dimensional representation vector, $z_t \in \mathbb{R}^k$, to a high-dimensional image as $x_t = G_\gamma(z_t)$. Thus, our goal of video recovery is equivalent to solving the

following optimization problem over z_t:

$$y_t = A_t G_\gamma(z_t) + e_t, \tag{3}$$

which can be viewed as a nonlinear inverse problem. Below we discuss three different methods for solving this inverse problem.

(a) *Latent code optimization:* fixed γ, update z_t.
(b) *Joint latent code and generator optimization:* update γ, z_t
(c) *Joint optimization with lowrank constraints:* update both γ, z_t with additional low-rank constraints on z_t.

A. Latent Code Optimization

In latent code optimization, we assume that the function $G_\gamma(\cdot)$ approximates the distribution of the set of natural images that contains our desired image. Thus, we can restrict our search for the underlying video sequence, x_t, within the range of the generator. In other words, we fix the network parameters, γ, and update only the latent codes, z_t. This is the same problem studied in [**?**] for image compressive sensing using generative models.

Given a pretrained generator, G_γ, measurement sequence, y_t, and the measurement matrices, A_t, we solve the following optimization problem to recover the low-dimensional latent codes:

$$\underset{z_1,\ldots,z_T}{\text{minimize}} \sum_{t=1}^{T} \|y_t - A_t G_\gamma(z_t)\|_2^2. \tag{4}$$

The reconstructed video sequence can be computed as $\hat{x}_t = G_\gamma(\hat{z}_t)$, where $\hat{z}_1, \ldots \hat{z}_T$ denote the solution of the problem in (4).

To solve the problem in (4), we use a gradient descent approach by foward- and back-propagating the gradient w.r.t. z_t through the fixed generator network.

The latent code optimization in (4) can solve the compressive sensing problem with high probability if the solution belongs to the range of the generator [**?**]. Otherwise, its solution is a poor estimate of the original image. Since the range of natural images is very large, and it is difficult to represent all of them with a single or a few generators, the latent code optimization application is limited to the case when a pretrained generator is available.

B. Joint Latent Codes and Generator Optimization

To jointly optimize the latent codes and generator parameters, we use the same formulation as in (4) but optimize it over the z_t and γ. The resulting optimization problem can be written as

$$\underset{z_1,\ldots,z_T;\gamma}{\text{minimize}} \sum_{t=1}^{T} \|y_t - A_t G_\gamma(z_t)\|_2^2. \tag{5}$$

The reconstructed video sequence can be generated using the estimated latent codes $(\hat{z}_1, \ldots, \hat{z}_T)$ and generator weights $(\hat{\gamma})$ as $\hat{x}_t = G_{\hat{\gamma}}(\hat{z}_t)$.

The joint optimization of latent code and network parameters offers the optimization problem a lot of flexibility to generate a wide range of images. We initialize latent codes with samples drawn from a Gaussian distribution and normalize them to have unit norm. We initialize γ with random weights using the initialization scheme in [**?**]. Initilizing the generator with a pretrained set of weights can potentially serve as a good initialization and lead to good and faster convergence. We test both variants, but observe little difference in performance; therefore, we use random initialization of parameters in this paper. Each iteration of joint optimization consists of two steps: 1) latent code optimization and 2) network parameter optimization. After every gradient descent update of the latent codes, z_t, we update the model parameters with stochastic gradient descent. In all of our experiments with joint optimization, we learn a single set of network weights for the entire sequence. We note that it is possible to divide a longer video sequences into small segments and learn different sets of network weights for each of them. At the end of our reconstruction process, we have a single set of trained weights $\hat{\gamma}$, reconstructed frames $\hat{x}_t$ and their corresponding optimal latent codes $\hat{z}_t$.

The range of any generator is quite limited and presumably depends on the types of images used during training. To highlight this limitation, we perform an experiment to reconstruct a video sequence from its masked version where 80% of the pixels are randomly missing under three different scenarios. The results are summarized in Figure 3 using four video sequences: '*Handwaving*' and '*Handclapping*' sequences from KTH video dataset and '*Archery*' and '*Apply Eye Makeup*' sequence from UCF101 video dataset. We center, crop, and resize all the frames to 64×64 pixels. We only select the first 32 frames of the entire video sequence for testing reconstruction performance. We show the reconstruction under three different scenarios:

(a) In the first experiment, we train a generator using all but the first 32 frames of the corresponding video sequences that we call *Generator1*. Then we used *Generator1* as a prior for the reconstruction of the 32 test frames from their masked measurements. Since the training and test frames belong to the same video sequence and share lot of similarities, we can recover the test frames using *Generator1* in (4).
(b) In the second experiment, we use a generator pretrained on CIFAR10 dataset that we call *Generator2*. We reconstruct the test frames using latent code optimization with *Generator2* as a prior. As CIFAR10 contains images from diverse categories, the pretrained generator should have some generalization but it cannot reconstruct the test frames with good quality.
(c) In the third experiment, we initialize the generator with a random set of weights using the initialization technique in [**?**] and jointly optimize the latent codes and network parameters. As we can observe from Figure 3, joint optimization with random initialization provides similar or better reconstruction quality than the latent code optimization with network pretrained on the target class of images.

The latent code optimization results presented in Figure 3 should not be surprising for the following reasons: We are providing a measurements y_t of a video sequence to the generator $G_\gamma(z_t)$ that has k degrees of freedom for each z_t; therefore, the range of sequences that can be generated by changing the z_t is quite limited for a fixed γ. The surprising thing, however, is that we can also recover quality images by

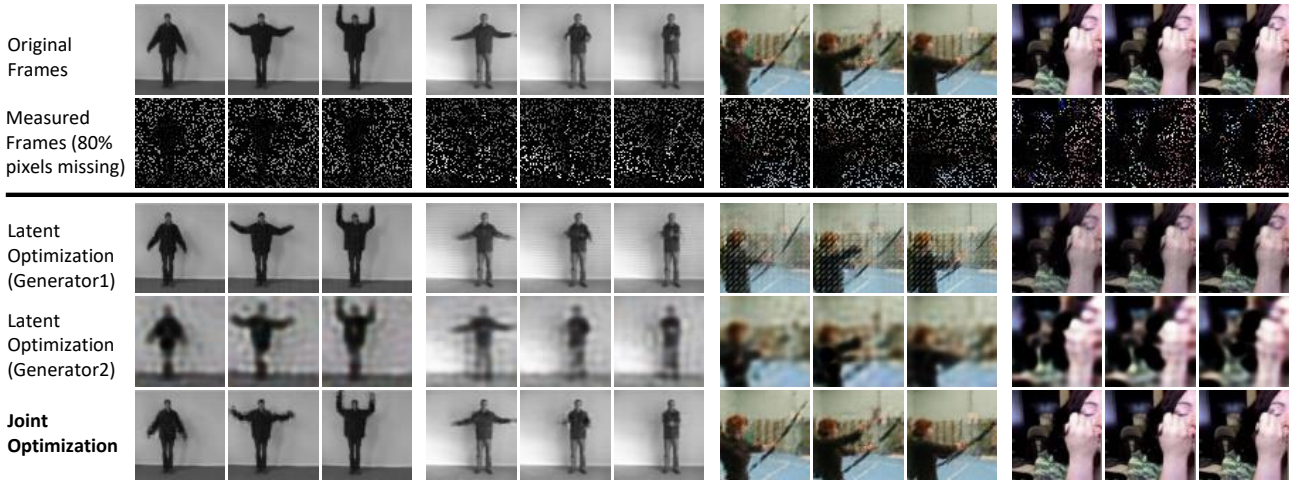


Fig. 3: Joint optimization versus latent code optimization. First row is the true images of the videos sequences. The second row contains the masked samples of the sequences. In the third row, we reconstruct frames with latent code optimization using a generator trained on some other frames of the same video sequence (Generator1). In the fourth row, we use latent code optimization with a generator trained on CIFAR10 dataset (Generator2). The fourth row is the reconstruction with joint optimization of generator initialized with random weights. We can observe that latent code optimization does not perform well (row 4) when we do not have generator pretrained on similar distribution. However, joint optimization performs as good as as or better than latent code optimization without any pretrained weights.

jointly optimizing the latent codes z_t and network weights γ while solving the compressive sensing problem. If we let γ change while we learn the z_t, then the network can potentially generate any image in $\mathbb{R}^n$ because the network has very large degrees of freedom. Note that in our generator, the number of parameters in γ is significantly larger than the size of x_t, y_t or z_t. In other words, we can overcome the range limitation of the generator by optimizing network parameters alongside latent code to get a good reconstruction from compressive measurements as well as good representative latent codes for the video sequence even though the network is highly overparameterized.

C. Low Rank Constraint

As we optimize over the latent codes and the network weights in joint optimization, the latent codes capture the temporal similarity of the video frames. To further exploit the redundancies in a video sequence, we assume that the variation in the sequence of images are localized and the latent codes sequence can be represented in a low-dimensional space compared to their ambient dimension. Let us define a matrix Z with all the latent codes as

$$Z = [z_1 \; z_2 \; \dots \; z_T],$$

where z_t is the latent code corresponding to t^{th} image of the sequence. To impose a low-rank constraint, we solve the following constrained optimization:

$$\underset{z_1,\dots,z_T;\gamma}{\text{minimize}} \; \sum_{t=1}^{T} \|y_t - A_t G_\gamma(z_t)\|_2^2 \;\; \text{s.t.} \;\; \text{rank}(Z) = r. \quad (6)$$

We solve (6) using a projected gradient descent method in which we project the latent code estimates after every iteration to a manifold of rank-r matrices. To do that, we compute Z matrix and its rank-r approximation using principal component analysis (PCA) or singular value decomposition (SVD).

In this manner, we can express each of the latent codes in terms of r orthogonal basis vectors $u_1, \dots, u_r$ as

$$z_i = \sum_{j=1}^{r} \alpha_{ij} u_j \quad (7)$$

where α_{ij} is the weight of the corresponding basis vector. We can represent a video sequence with T frames with r orthogonal codes, and the lowrank representation of latent codes requires $r \times k + r \times T$ parameters compared to $T \times k$. This offers $r(\frac{1}{T} + \frac{1}{k})$ times compression to our latent code representation. As we observe later, we use $r = 4$ for $k = 256$ and $T = 32$ which gives us compression of 0.14 in latent code representation.

Algorithm 1 Generative Models for Low Rank Representation and Recovery of Videos

Input: Measurements y_t, measurement matrices A_t, A generator structure $G_\gamma(\cdot)$
Initialize the latent codes z_t and generator weights γ randomly and normalize z_t with its 2-norm.
repeat
Compute gradients w.r.t. z_t via backpropagation.
Update latent code matrix $Z = [z_1 \; \cdots \; z_T]$.
Truncate Z to a rank-r matrix via SVD or PCA.
Compute gradients w.r.t. γ via backpropagation.
Update network weights γ.
until convergence or maximum epochs
Output: Latent codes: $z_1, \dots, z_T$ and network weights: γ

III. Experimental Setup

In this section, we describe our experimental setup and empirical results. We focus our experiments on three different compressive sensing problems: denoising, inpainting, and

TABLE I: Reconstruction performance measured in terms of PSNR for different compressive sensing problems. We show comparison with TVAL3D (3D extension of TVAL3 [**?**]) and deep decoder [**?**]. The results are averaged over five experiments with different random measurement matrices (or noise in the case of denoising).

Video Sequence	Rotating MNIST	Handclapping	Handwaving	Walking	Apply Eye Makeup	Archery	Band Marching
Denoising for additive Gaussian noise of 20dB SNR							
TVAL3D	35.8	32.2	30.4	30.5	34.5	31.5	30.6
UP Deep Decoder	28.9	28.4	25.6	28.3	28.1	29.6	28.1
OP Deep Decoder	36.6	31.1	30	31	34.4	**33**	31.6
Joint Optimization	**36.9**	**32.7**	30.7	**31.2**	36.1	32.1	31.3
Joint Opt + Low Rank	36.8	32.3	**30.8**	30.7	**36.4**	32	**31.7**
Inpainting with 80% pixels randomly missing							
TVAL3D	21.1	29.2	23.4	24.5	28.2	27.1	24.8
UP Deep Decoder	25.5	26.5	23.3	26.3	27.2	29	23.3
OP Deep Decoder	30.1	30.2	26.7	27.9	32.4	**32.5**	26.2
Joint Optimization	29.3	**34.9**	**28.1**	**28.9**	35.8	32	26.8
Joint Opt + Low Rank	**29.5**	34.3	27.3	27.8	**36.6**	30.4	**27.6**
Spatial compressive sensing with compression rate = 0.2							
TVAL3D	29.8	32.1	28.9	28	33.9	28.4	27.8
UP Deep Decoder	30	27	24.9	26.7	26.2	27.6	22.5
OP Deep Decoder	35.2	32.9	**30.6**	29	33.1	**31.2**	27.4
Joint Optimization	35.3	**35.6**	29.7	28.9	**36**	29.3	27.8
Joint Opt + Low Rank	**35.4**	34.7	29	**29.1**	35.9	28.8	**29.1**

spatial compression by random projection. We also show some empirical results for coded flutter shutter problem where our algorithm is especially suitable. For a video sequence of T frames, we generate T independent measurement matrices. For color images, we use the same measurement matrix for each color channels. The total number of frames in each video sequence is 32, unless stated otherwise. For the low-rank constraint, we select the mean of the latent matrix Z and top 3 principal components (i.e., we need 4 vectors to represent the entire video sequence instead of 32.)

Choice of generator. We use the well-known DCGAN architecture [**?**] for our generators, except that we do not use any batch-normalization layer because gradient through the batch-normalization layer is dependent on the batch size and the distribution of the batch. As shown in Figure 1, in DCGAN generator framework, we project the latent code, z, to a larger vector using a fully connected network and then reshape it so that it can work as an input for the subsequent deconvolutional layers. Instead of using any pooling layers, the DCGAN architecture uses strided convolution [**?**]. All the intermediate strided convolution layers are followed by ReLU activation. The last strided convolution layer is followed by Tanh activation function to generate the reconstructed image $x = G(z)$. In our experiment, we use videos of different resolutions. To generate those videos, we use different generators following the DCGAN framework. In Table II, we report the detailed structure of the generators we use in the experiments.

The latent code dimension for grayscale 64×64 video sequence is 64. The latent code dimension for color 64×64 video sequence is 256. The latent code dimension for 256×256 video sequence is 512. We use Adam optimizer for generator weights optimization and SGD for latent code optimization. The learning rate for latent code optimization was 10. We use ADAM optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ for network parameters optimization. The initial learning rate for network parameter optimization was 0.0025. We decay the learning rate for network parameter by 25% every 500 iterations.

Comparison with existing methods. We show comparison with classical total variation minimization based TVAL3D (3D extension of TVAL3 [**?**]) algorithm and generative prior based deep decoder [**?**] algorithm. As we mentioned earlier, deep decoder does not optimize latent code, rather it uses fixed latent codes which are drawn from Gaussian distribution. We use deep decoder implementation from [**?**] and TVAL3D implementation from [**?**].

We use two different deep decoder settings: underparameterized deep decoder (UP deep decoder) and overparameterized deep decoder (OP deepdecoder). The UP deep decoder was proposed in the original deep decoder paper [**?**], but we also report the results for OP deep decoder because it shows better performance. We use default 6 layer architecture of deep decoder. In the UP deep decoder, the number of parameters in UP deep decoder is 11,304 and 11,288 for RGB and grayscale images, respectively. The number of parameters in OP deep decoder is 397,056 and 396,544 for RGB and grayscale images, respectively. We need separate generator for every frame which increases the effective number of parameters for the video sequence by a factor T for T frames. As T=32 for most of the experiments reported in the paper, the effective number of parameters for OP deep decoder is 12,705,792 and 12,689,408 for RGB and grayscale images, respectively. We report the qualitative reconstruction results for OP deep decoder only because quantitative reconstruction results for UP deep decoder are significantly worse. This effect is also recently observed in [**?**].

We also show some comparison with video extension of deep image prior [**?**] algorithm. We discuss details of this approach later in the paper.

Video datasets. We test all the methods on different synthetic and real video sequences. In this paper we report the results for one synthetic sequence which we refer to as '*Rotating MNIST*'. In this sequence, we resize one MNIST digit to 64×64 and rotate by 2° per frame for a total of 32 frames. We experiment on different real video sequences

TABLE II: Generator structures and corresponding number of parameters for different image sizes. $h \times w \times c$ denote height , weight, and color channels, respectively.

	Network Parameters		
Output size	$\mathbf{64 \times 64}$	$\mathbf{64 \times 64 \times 3}$	$\mathbf{256 \times 256 \times 3}$
FC + ReLU	524,288	2,097,152	4,194,304
Conv 1+ ReLU	2,097,152	2,097,152	2,097,152
Conv 2+ ReLU	524,288	524,288	524,288
Conv 3+ ReLU	131,072	131,072	131,072
Conv 4+ Tanh/ReLU or ReLU	1,024	3,072	32,768
Conv 5+ ReLU	-	-	8,192
Conv 5+ Tanh	-	-	768
Total # params	**3,277,824**	**4,852,736**	**6,988,544**

from publicly available KTH human action video dataset [**?**] and UCF101 dataset [**?**]. In Table I, we report our results for '*Handclapping*', '*Handwaving*' and '*Walking*' video sequences from KTH dataset; '*Archery*', '*Apply Eye Makeup*' and '*Band Marching*' video sequences from UCF101 dataset. We center and resize every frame in KTH videos to 64×64 and UCF101 videos to 256×256 pixels.

Performance metric. We measure the performance of our recovery algorithms in terms of the reconstruction error PSNR. For a given image x and its reconstruction $\hat{x}$, PSNR is defined as

$$\text{PSNR}(x, \hat{x}) = 20 \log_{10} \frac{\max(x) - \min(x)}{\sqrt{\text{MSE}(x, \hat{x})}}$$

where $\max(x)$ and $\min(x)$ are the maximal and minimal values in x, respectively, and MSE is the mean squared error. Unless otherwise stated, all the results are averaged over 5 experiments using different measurement matrices or noise.

IV. Results and Analysis

A. Sequence Size vs Performance

To evaluate the effect of sequence size on the performance of our method, we perform joint optimization experiments with video sequences of different sizes. We report our results for three different video sequences in Figure 4. We consider three different tasks. The first task is video approximation, where we approximate the original video sequences using a generator (i.e., A_t is an identity matrix for all t). We observe that as we increase the size of the video sequence, the quality of approximated video sequences degrades. This intuitively makes sense because a network with sufficient complexity should be able to approximate a single image perfectly. However, as we increase the size of the video sequence while keeping the same network structure, our algorithm has to find a new set of optimal weights that can generate the entire sequence. The reduction in reconstruction performance is more pronounced when every frame of the video is different. Our second task is image inpainting with 80% randomly missing pixels and the third task is compressive sensing with 20% available measurements. In both cases, we have far fewer number of measurements available than that in the approximation task. As the consecutive frames of a video sequence are close to each other, the increased size of the video sequence actually helps by providing more effective measurements to the generator. However, the generator capacity still remains a barrier. On one hand, we have more (diverse) measurements available while optimizing over a larger video sequence, which can provide a gain in the performance with a longer sequence. On the other hand, we have to find a set of network parameters that can generate all the (diverse) frames at once, which can cause a loss in performance with longer sequence. We observe in our experiments that the recovery performance increases with the length of the video sequence up to a certain point and then it saturates. We select the size of the video sequences as 32 for our next experiments based on these results.

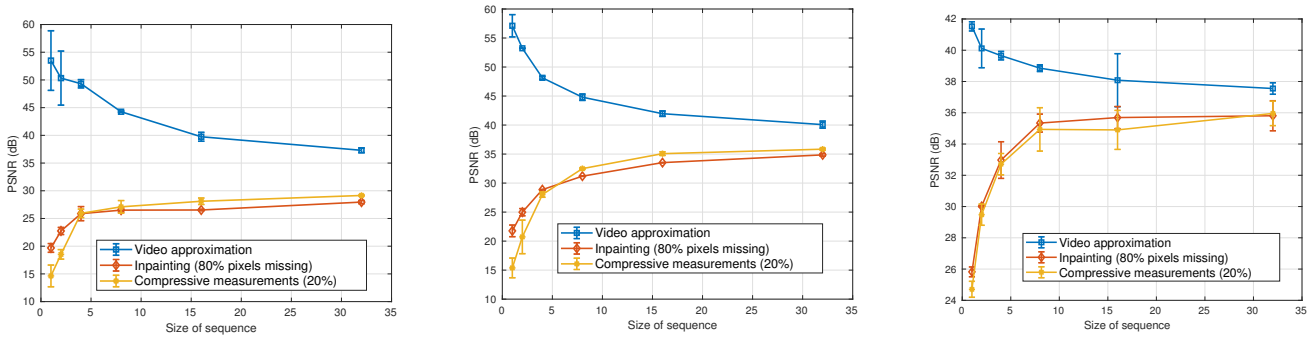
B. Denoising

We first explore the potential of joint optimization on the denoising problem. In our denoising setup, the measurement matrix is identity and the noise, e_t, is drawn from zero mean Gaussian distribution. We report the reconstruction results of different video sequences for different algorithms in Table I. We observe that joint optimization performs significantly better than UP deep decoder and provides similar results as TVAL3D and OP deep decoder. Note that we do not optimize over latent code for deep decoder. We also need to train a separate network for each frame, which requires huge computational power and memory. We report the memory and computational complexity comparison in Table III. We also observe that joint optimization with low-rank constraint provides similar performance.

We present some denoising results for different techniques with additive Gaussian noise at 20dB SNR on different sequences in Figure 5. The performance curves in terms of average PSNR over a range of SNR levels are presented in Figure 6. We observe that reconstruction performance of joint optimization is better than classical TVAL3D. For large noise, joint optimization shows better or comparable performance with the deep decoder. However, for small noise, deep decoder seems to outperform joint optimization. In the case of deep decoder, we learn a separate network for every frame, and as we observe in the previous section that for a fixed generator, image approximation performance is better for single image. In the low noise regime, denoising problem is almost same as an approximation problem. In the case of joint optimization, we are learning a single set of network parameters for the entire video sequence; therefore, joint optimization has a limitation due to the representation capacity of the generator network. We can also observe in Figure 6 that the curves corresponding to UP deep decoder is flat, which is because of the fact that the UP deep decoder has a certain representation capacity, and once that capacity is reached for a single frame the results do not improve even if the noise level decreases.

C. Inpainting

Our second experiment is on inpainting problem where we randomly drop a fraction of the pixels from each frame and reconstruct the original video sequence from available pixels. We report the results for 80% missing pixels in Table I. We observe that reconstruction performance of joint optimization is significantly better than classical TVAL3D and deep decoder. We also show some reconstructions of different video sequences from 20% available pixels (80% pixels are randomly missing) in Figure 7. From this figure, we can observe that even though the reconstruction results of deep decoder is similar to joint optimization in terms of PSNR, deep decoder fails to reconstruct high frequency details reliably (see *Archery* results).



(a) Handwaving **(b)** Handclapping **(c)** Apply Eye Makeup

Fig. 4: Sequence size vs performance for video approximation and compressive sensing tasks. Here the results corresponds to joint optimization. We can observe that increasing video length improves compressive sensing performance for joint optimization. This effect diminishes with the increased size of video sequences.

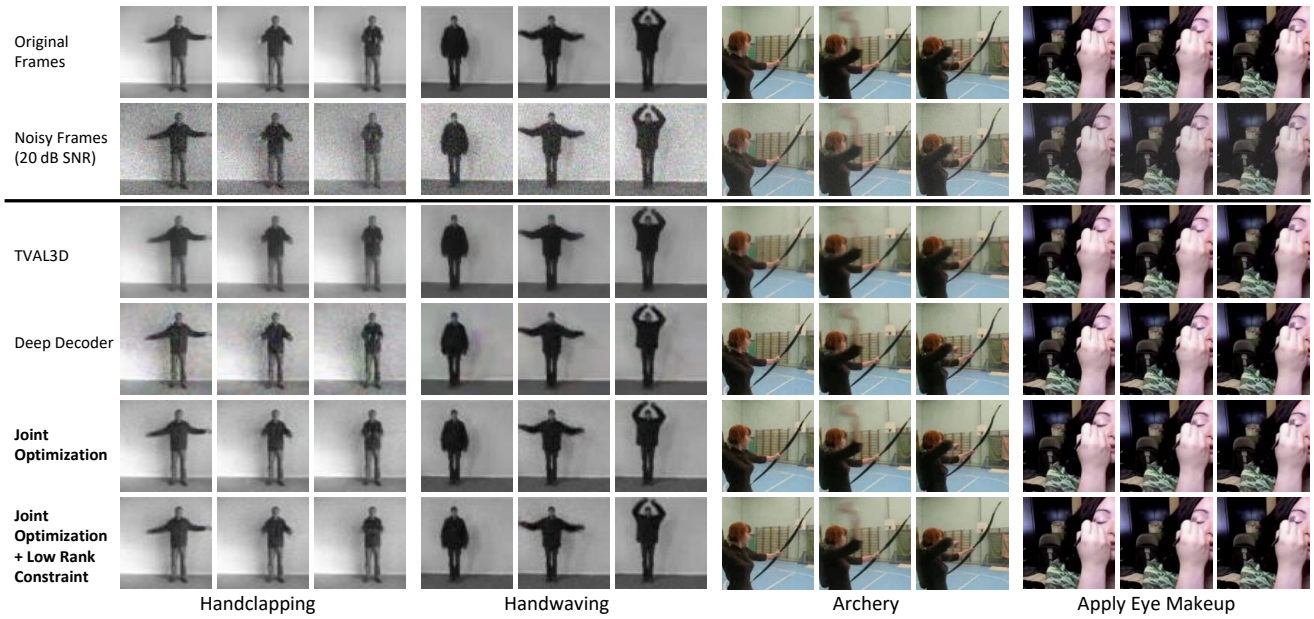
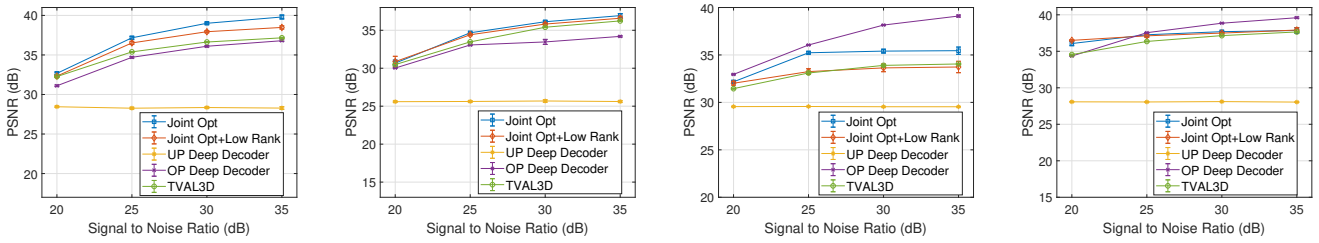


Fig. 5: Reconstruction of different video sequences using different algorithms for denoising problem. Handclapping and Handwaving video sequences are 64×64 and Archery and Apply Eye Makeup video sequences are 256×256. The error bars are standard deviation intervals. The deep decoder reconstruction here correspond to overparameterized deep decoder structure. All the comparing algorithms show very good reconstruction quality.



(a) Handclapping **(b)** Handwaving **(c)** Archery **(d)** Apply Eye Makeup

Fig. 6: Reconstruction quality curves for denoising experiments with different algorithms for different levels of signal to noise ratio. The curves also show standard deviation intervals. We compare the performance for (a) Handclapping (b) Handwaving (c) Archery (d) Apply Eye Makeup video sequences. All the comparing methods other than UP deep decoder performs similarly. The curves suggest that UP deep decoder has reached its limit to generate the sequences.

Since we optimize the generator using a number of frames in joint optimization, the missing information in one frame may be available in a neighboring frame, and that can potentially help the joint optimization perform better in reconstructing the high frequency details as shown in Figure 7.

We also show inpainting performance for different fractions

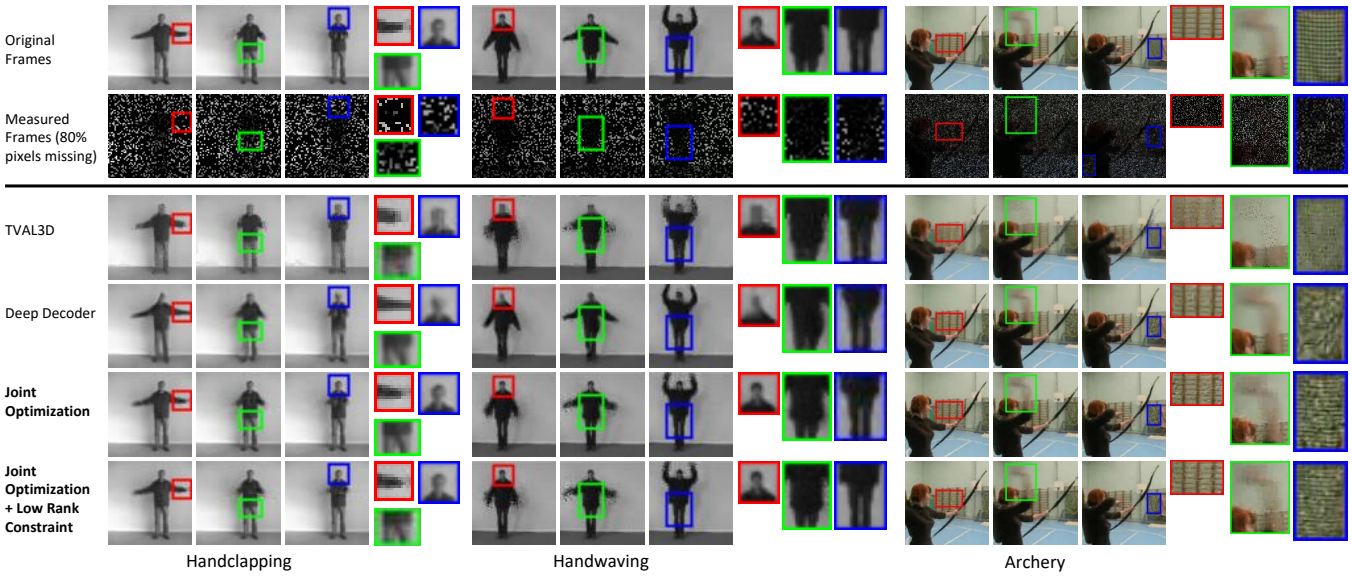
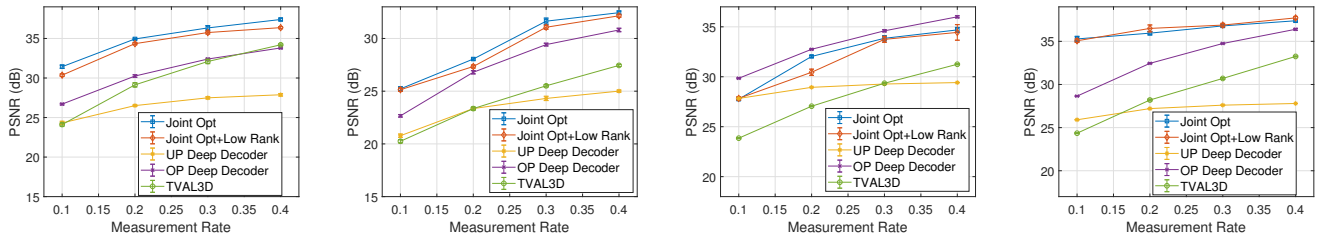


Fig. 7: Some reconstruction results on inpainting problem. Handclapping and Handwaving video sequences are 64×64 and Archery sequence is 256×256. The deep decoder reconstruction here correspond to overparameterized deep decoder structure. The boxed regions are zoomed for details. We can observe that joint optimization gives better reconstruction than the comparing algorithms in terms of details.



(a) Handclapping **(b)** Handwaving **(c)** Archery **(d)** Apply Eye Makeup

Fig. 8: Inpainting performance for different available measurement rate for (a) Handclapping (b) Handwaving (c) Archery (d) Apply Eye Makeup video sequences. Measurement rate represents the available fraction of the total pixels. The error bars are standard deviation intervals. Other than Archery sequence, joint optimization outperforms the other comparing methods especially at lower measurement rate.

of missing pixels in Figure 8. From the comparison with the other algorithms shown in Figure 8, we can observe that joint optimization with/without low rank constraint outperforms other comparing algorithms especially when we have very few number of measurements available.

D. Compressive Sensing

In this section, we discuss our experiments on recovery of frames from their compressive random projections. In these experiments, we use separable measurements, $Y = PXQ$, where X, Y are reshaped versions of x, y as 2D matrices, P and Q are left and right random projection matrix. In our experiment, we use $P = Q^T$, and select their size so that the total number of measurements in Y is m. We draw each sample of P from $N(0, \frac{1}{\sqrt{m}})$ distribution.

We summarize the results for this experiment in Table I. We select $m = 29 \times 29$ for 64×64 images and $m = 114 \times 114$ for 256×256 images, which gives us a compression of factor of approximately 20%. We observe from Table I that joint optimization with and without low-rank constraint slightly outperforms TVAL3D. It performs similarly as deep decoder with much lower memory requirement and computational complexity.

We show some reconstructions for compressive sensing with 20% compressive measurements in Figure 9. We can observe that he reconstructions are comparable with other algorithms. We also show reconstruction performance for different compression ratio in Figure 10. We can observe from Figure 10 that joint optimization with or without low rank constraint outperforms TVAL3D and UP deep decoder. However, it performs at par with if not better than OP deep decoder.

E. Flutter Shutter

We also perform an experiment with a computational photography problem known as coded flutter shutter [**?**], [**?**], [**?**], [**?**], [**?**] in which a low-speed camera is used to capture a modulated high-speed video. A single observed frame can be modeled as coded and multiplexed version of a number of frames in the sequence. Our goal is to recover the individual frames from the multiplexed frame. Mathematically, we can

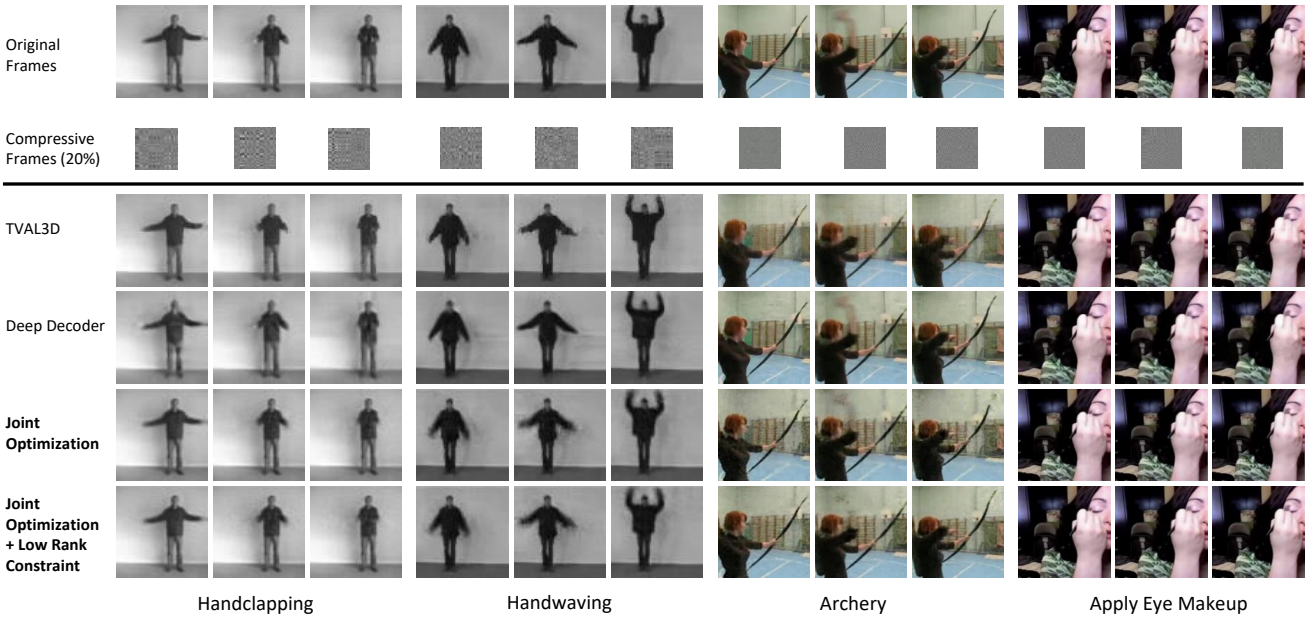
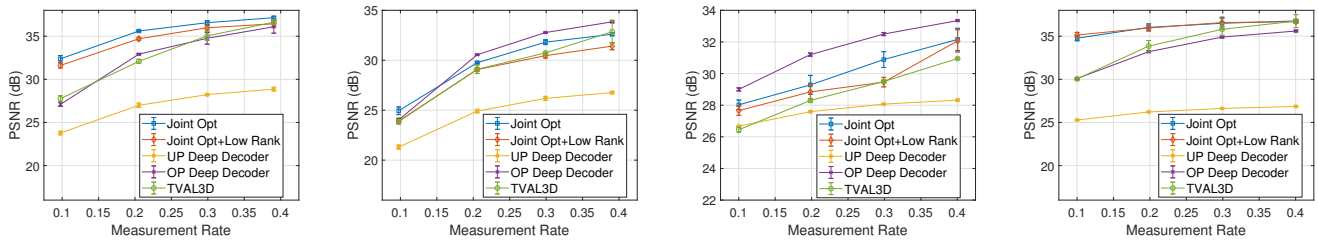


Fig. 9: Some reconstruction results on spatial compressive sensing problem. Handclapping and Handwaving video sequences are 64×64 and Archery and Apply Eye Makeup video sequences are 256×256. The compressive frames from Handclapping and Handwaving are 29×29 whereas the compressive frames from Archery and Apply Eye Makeup video sequences are 114×114. The deep decoder reconstruction here correspond to overparameterized deep decoder structure. We can observe that the reconstructions are similar for the comparing algorithms.



(a) Handclapping **(b)** Handwaving **(c)** Archery **(d)** Apply Eye Makeup

Fig. 10: Compressive sensing performance for different available measurement rate for (a) Handclapping (b) Handwaving (c) Archery (d) Apply Eye Makeup video sequences. Measurement rate (or compression ratio) represents the available fraction of the total measurements. The error bars are standard deviation intervals. We can observe from the curves that joint optimization performs at par with the other comparing methods.

formulate the problem as

$$y_p = \sum_{t=Mp}^{Mp+M-1} A_t x_t + e_p, \quad \text{for } p = 1, \ldots, T/M, \qquad (8)$$

where we observe a single measurement frame for every M consecutive frames. Thus, we have $\frac{T}{M}$ measurement frames for the entire video sequence. We choose A_i in a similar manner as the inpainting mask (i.e., 50% pixels are randomly missing). Our joint optimization can solve this problem because we can jointly estimate multiple frames while solving a single optimization problem. In contrast, if we train a separate network for every single frame (as done in DIP and deep decoder), the recovery problem will not be as straightforward. To estimate the video sequence from coded, multiplexed measurements, we solve the following recovery problem:

$$\underset{z_1,\ldots,z_T;\gamma}{\text{minimize}} \sum_{p=1}^{T/M} \|y_p - \sum_{t=Mp}^{Mp+M-1} A_t G_\gamma(z_t)\|_.^2 \qquad (9)$$

We present some reconstructed images for coded flutter shutter in Figure 11. Because of the high memory and computational requirements, deep decoder is not suitable for this problem. We present a comparison with TVAL3D reconstruction. Joint optimization with and without low-rank constraints successfully recover fast motion that TVAL3D fails to recover.

F. Rank of the Latent Matrix

In this section, we evaluate the performance of joint optimization with low-rank constraints for different choice of the rank. We perform inpainting experiment with 80% missing pixels using different values of rank. We plot the reconstruction PSNR performance curves for different video sequences in Figure 12. Rank-1 corresponds to using a fixed (mean) vector as the latent code for all the frames, which would reconstruct the same frame for the entire sequence. As we increase the rank of the latent code matrix, we observe that reconstruction quality improves and rank-4 reconstruction gives us a good

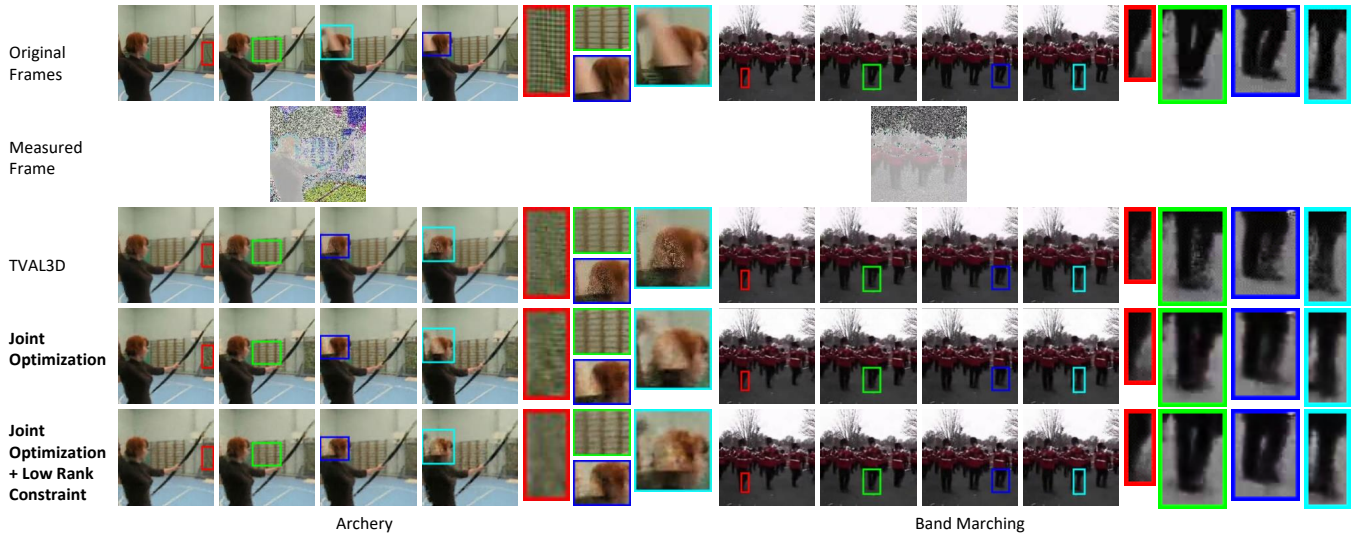


Fig. 11: Some reconstructions for flutter shutter problem. Here we have a single measurements for every 4 non overlapping frames. We can observe that TVAL3D suffers ghosting effect for the fast changing parts of the videos such as the hand or leg movement. However, they perform similarly in background details reconstruction.

performance for all the sequences. Note that for rank-4, we select the mean vector and top-3 principal components to represent the entire sequence with 32 frames.

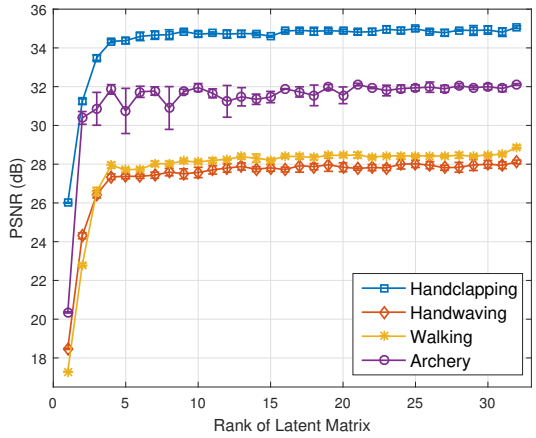


Fig. 12: Effect of different value of rank for low rank constraint in inpainting problem with 80% pixels randomly missing. We also show standard deviation interval for each point.

G. Computational Complexity

The computational complexity of our proposed methods vary with the choice of the generator structure. We have chosen DCGAN generator structure for our experiments. We compare the computational complexity of our algorithm with UP and OP deep decoder [**?**]. The memory requirement mentioned here is for a single frame. The memory requirement is calculated using `torchsummary` package [**?**]. The time consumption is recorded for the inpainting of RGB video sequences with 32 frames from 80% missing pixels. We report average time consumption over 5 experiments. The number of iterations, measurement matrix and the videos sequences of the corresponding size were kept the same. The experiments for this comparison were run on the same CPU equipped with Nvidia Titan Xp GPU.

From the memory requirement and time consumption, it is evident that joint optimization is much less complex and consumes much less memory compared to OP deep decoder. Although the memory requirement and complexity of UP deep decoder comes close to that of joint optimization, UP deep decoder reconstruction performance is poor (Table I, Figure 6,8,10).

TABLE III: Comparison of joint optimization with DCGAN and deep decoder in terms of computational complexity and memory requirement. The memory requirement is for each frame reconstruction. The average time consumption is calculated for video sequences with 32 frames.

Size	$\mathbf{64 \times 64}$	$\mathbf{256 \times 256}$
Memory Requirement (Forward and Backpropagation)		
UP Deep decoder	2.75 MB	44.03 MB
OP Deep decoder	66.48 MB	1239.75 MB
Joint Opt with DCGAN	2.06 MB	10.88 MB
Average time consumed (Forward and Backpropagation)		
UP Deep decoder	120 sec	710 sec
OP Deep decoder	180 sec	3042 sec
Joint Opt with DCGAN	14.2 sec	203 sec

H. Comparison with Video DIP

In section IV-A, we have demonstrated that optimizing over a video sequence improves reconstruction performance. However, as we have mentioned before, DIP [**?**] trains one network per image which puts it at a disadvantage while comparing with joint optimization. So, we made an extension of DIP for video sequence and refer to it as "Video DIP". In this approach, we draw entries in the latent matrix Z from a Gaussian distribution and keep it fixed as we solve the following optimization:

$$\hat{\gamma} = \arg\min_{\gamma} \sum_{t=1}^{T} \|y_t - A_t G_\gamma(z_t)\|_2^2 \tag{10}$$

TABLE IV: Effect of initial latent matrix for different inverse problems. We have drawn latent matrix in way that the initial latent codes form a line. The results are averaged over fifteen experiments with five different random measurement matrices and three different initializations. We use same measurement matrices and initializations for both approaches.

	Rotating MNIST		**Handclapping**		**Handwaving**		**Walking**		**Apply Eye Makeup**		**Archery**		**Band Marching**	
	Video DIP	**Joint Opt**	**Video DIP**	**Joint Opt**	**Video DIP**	**Joint Opt**	**Video DIP**	**Joint Opt**	**Video DIP**	**Joint Opt**	**Video DIP**	**Joint Opt**	**Video DIP**	**Joint Opt**
Inpainting (80% missing)	28.6	30.1	31	34.1	23	26.8	23.8	26.5	34.8	37	30.5	31.9	28.8	29
Compressive Measurements (20%)	33.8	35.5	33.1	35.5	24.6	30.1	23.9	28.6	32.9	36.1	29.2	29.9	28.3	29.3

TABLE V: Performance analysis between Video DIP and joint optimization when all the frames in the video sequence are not close to each other. The results are averaged over twelve experiments with four different random measurement matrices and three different initializations. We use same measurement matrices and initializations for both approaches.

	Handclapping + Handwaving		**Handclapping + Walking**		**Handwaving + Walking**	
	Video DIP	**Joint Opt**	**Video DIP**	**Joint Opt**	**Video DIP**	**Joint Opt**
Inpainting (80% missing)	31.5	33.3	32	33	27.9	29.4
Compressive Measurements (20%)	29	32.7	29.4	32.4	26.5	29.6

We use the same architecture for Video DIP as we use for joint optimization.

We observe that even if we extend DIP for video sequence, it still suffers from two main drawbacks that we mentioned earlier. First drawback is the dependence on the initialization of latent matrix as it remains fixed. If the initialization is *bad*, Video DIP will fail to provide good reconstruction. We demonstrate this effect with an example. We force the latent codes to fall on a line by fixing two of the latent codes z_1 and z_T, each drawn from $N(0,1)$ and initialize the other latent codes by linear interpolating between z_1, z_T. We expect Video DIP to perform worse in this case because we are forcing the network to map frames to a line in a 2D plane whereas the videos contain much complex motions. We report the reconstruction results for both Video DIP and joint optimization in Table IV. We experiment on inpainting and compressive sensing with 20% available measurements. Joint optimization performs better than Video DIP in all the cases.

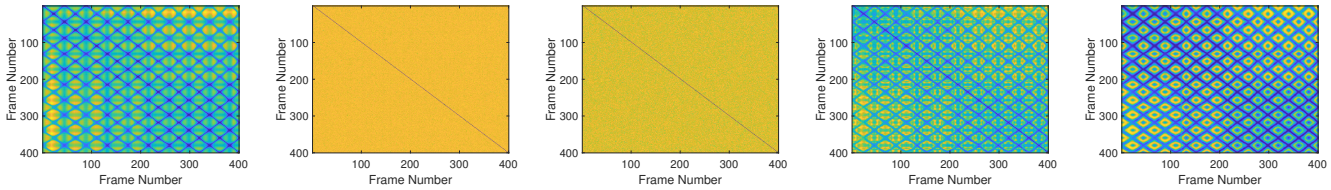
The second drawback is that Video DIP does not assume or retain any similarity structure in the latent code representation. This will in turn affect the reconstruction quality if the frames in the videos are very different from one another. To demonstrate this effect, we create a video sequence with 64 frames temporally concatenating 32 frames from two different sequences one after another. In Table V, we report average results of four experiments with different measurement matrices. We perform experiments for inpainting and compressive sensing problems. For both Video DIP and joint optimization, latent matrices are initialized with elements drawn from $N(0,1)$ distribution. Since we initialize the latent matrix at random, it is possible that similar latent codes are assigned to frames that are quite different. As the latent codes are fixed in Video DIP, finding the network parameters that map very different frames to latent codes that are very similar can be a challenging task. In contrast, joint optimization method updates both the latent codes and network parameters; therefore, it can adjust the latent codes so that similar latent codes are mapped to similar frames and vice versa. From Table V, it is evident that joint optimization is better suited for such videos as it outperforms Video DIP in each case.

The latent codes obtained from joint optimization also reserve the similarities among video frames. We demonstrate this property using a compressive sensing experiment, where we capture 20% measurements of each frame with an independent random matrix. We use 400 frames of '*Handwaving*' sequence for this experiment instead of 32 frames. In the sequence, the handwaving action is repeated multiple times, each of which takes around 45 frames. We compute a cosine similarity matrix between all the image pairs, which is plotted in Figure 13(a). Since compressive measurements are independent of one another, we do not expect any similarity between them, as seen in Figure 13(b). Latent codes in Video DIP are also independent and randomly selected, and they do not reflect the similarity structure of the video frames, as shown in Figure 13(c). We optimize the latent codes in joint optimization, and they preserve the similarity structure as we can observe from Figure 13(d). From Figure 13(e), we can observe that low rank constraint further enhances the similarity structure in latent code matrix.

To further investigate the similarity structure in the latent codes obtained by joint optimization, we perform another experiment in which we concatenate 16 frames from each of the six different video sequences ('*Handwaving*', '*Handclapping*', '*Walking*', '*Archery*', '*Apply Eye Makeup*', and '*Band Marching*', in the same order) to create a new sequence with 96 frames. We perform compressive sensing experiment on this video sequence with 20% measurements. The reconstruction PSNR for joint optimization is 29.12 dB, joint optimization with low-rank is 27.9 dB, and Video DIP is 26.4 dB. The cosine similarity matrices for the video frames, compressive measurements, latent codes for Video DIP, latent codes for joint optimization, and latent codes for joint optimization with low-rank are presented in Figure 14(a)–(e). We can distinguish the video sequences from the pairwise similarity matrices of the latent codes we estimate with joint optimization. We observe that the low-rank constraint improves the similarity matrix.

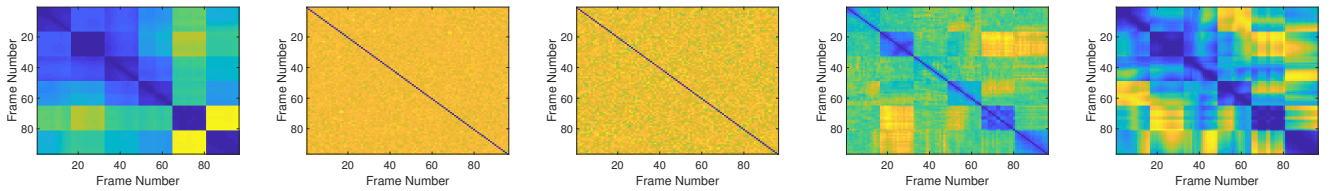
V. Conclusion

In all our experiments, we observe that joint optimization performs remarkably well for compressive measurements. Even though the number of measurements are extremely small



(a) Original Frames **(b)** Compressive Frames **(c)** Latent Matrix of Video DIP **(d)** Latent Matrix of Joint Optimization **(e)** Latent Matrix of Joint Optimization + Low Rank Constraint

Fig. 13: Pairwise cosine similarity between frames, measurements or latent codes for extended Handwaving video sequence where Handwaving action is repeated in an interval of around 45 frames. Blue indicates highest similarity whereas yellow indicates lowest similarity. We can observe that the similarity pattern in the original frames are not maintained in the compressive frames. As the Video DIP latent codes are drawn at random, we do not observe any similarity pattern in them (c). However, the corresponding latent matrix for joint optimization (d) captures the similarity structure. Low rank constraint (e) further enhances this similarity.



(a) Original Frames **(b)** Compressive Frames **(c)** Latent Matrix of Video DIP **(d)** Latent Matrix of Joint Optimization **(e)** Latent Matrix of Joint Optimization + Low Rank Constraint

Fig. 14: Pairwise cosine similarity between frames, measurements or latent codes for extended mixed video sequence where 16 frames of 6 different video sequences (Handwaving, Handclapping, Walking, Archery, Apply Eye Makeup, Band Marching in order) are concatenated in the temporal dimension. Blue indicates highest similarity whereas yellow indicates lowest similarity. We observe that adding low rank constraint further bolster the similarity observed in the frames of same video sequences found by joint optimization.

compared to the number of parameters in γ, the solution almost always converges to a good sequence. Introducing low-rank constraint in the optimization, we get additional degree of compression with comparable performance. We also show comparison with classical and generative prior based techniques in terms of reconstruction performance. Furthermore, we report comparison of computational complexity between joint optimization and deep decoder for video reconstruction. We also demonstrate one application of joint optimization in coded flutter shutter problem where its tractable memory requirement and lower computational cost makes it more suitable than other generative prior based approaches. We made an implementation of our algorithm available at https://github.com/CSIPlab/gmlr.

Rakib Hyder received the B.Sc. degree in electrical and electronic engineering from the Bangladesh University of Engineering and Technology, Dhaka, Bangladesh in 2016. He is currently pursuing Ph.D. degree at the University of California, Riverside, Riverside,CA, USA. His research interests include generative modeling, machine learning, computer vision, inverse problems, and signal processing.

M. Salman Asif received his B.Sc. degree in 2004 from the University of Engineering and Technology, Lahore, Pakistan, and the M.S.E.E degree in 2008, and the Ph.D. degree in 2013 from the Georgia Institute of Technology, Atlanta, GA, USA. He was a Senior Research Engineer at Samsung Research America, Dallas, TX, USA, from August 2012 to January 2014, and a Postdoctoral Researcher at Rice University from February 2014 to June 2016. He is currently an Assistant Professor in the Department of Electrical and Computer Engineering at the University of California, Riverside. He received Hershel M. Rich Outstanding Invention Award in 2016, UC Regents Faculty Fellowship Award in 2017, and Google Faculty Award in 2019. His research interests broadly lie in the areas of information processing and computational sensing with applications in signal processing, machine learning, and computational imaging.