Generative Models for Low-Dimensional Video Representation and Compressive Sensing

Anonymous

Abstract—Generative priors have become highly effective in solving inverse problems including denoising, inpainting, and reconstruction from few and noisy measurements. With a generative model we can represent an image with a much lower dimensional latent 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. However, recent studies revealed that even untrained deep neural networks can work as a prior for recovering natural images. These approaches update the network weights keeping latent codes fixed to reconstruct the target image from the given measurements. In this paper, we optimize over network weights and latent codes to use untrained generative network as prior for video compressive sensing problem. We show that by optimizing over latent code, we can additionally get concise representation of the frames which retain the structural similarity of the video frames. We also apply low-rank constraint on the latent codes to represent the video sequences in even lower dimensional latent space. We empirically show that our proposed methods provide better or comparable accuracy and low computational complexity compared to the existing methods.

I. INTRODUCTION

A. Motivation and Related Works

Compressive sensing refers to a broad class of problems in which we aim to recover a signal from a small number of measurements [1]–[3]. Suppose we are given a sequence of measurements for \( t = 1, \ldots, T \) as

\[
y_t = A_t x_t + e_t,
\]

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 \( (y_t) \) becomes very small compared to the number of unknowns \( (x_t) \).

Classical signal priors exploit sparse and low-rank structures in images and videos for their reconstruction [4]–[10]. However, the natural images exhibit far richer nonlinear structures than sparsity alone. We focus on a newly emerging generative priors that learn a function that maps vectors drawn from a certain distribution in a low-dimensional space into images in a high-dimensional space.

The generative model and optimization problems we use are inspired by recent work on using generative models for compressive sensing in [17]–[23]. Compressive sensing using generative models was introduced in [17], which used a trained deep generative network as a prior for image reconstruction from compressive measurements. Afterwards deep image prior (DIP) used an untrained convolutional generative model as a prior for solving inverse problems such as inpainting and denoising because of their tendency to generate natural images [22]; 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 [19], [20]. Both DIP [22] and deep decoder [20] update the model weights 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 observed 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. (2) Both of these methods train one network per image. A naive 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.

Therefore, we propose joint optimization over network weights \( \gamma \) and the latent codes \( z_t \) to reconstruct video sequence. Thus we learn a single generator and a set of latent codes to represent a video sequence. 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 types of representations we use in this paper are shown in Figure 1.

B. Our Contribution

In this paper, we reconstruct a video sequence from the compressive measurements in (1) by jointly optimizing over the latent codes \( z_t \) and the network parameters \( \gamma \). 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.

1. We demonstrate that joint optimization allows us to learn a single generator network for an entire video sequence and...
We initialize \( \gamma \)

To jointly optimize the latent codes and generator parameters

The reconstructed video sequence can be generated using the

Optimizing latent codes can only reconstruct images in the range of the

iteration of joint optimization consists of two steps: 1) latent

but observed little difference in performance; therefore, we

scheme in \[24\]. Initilizing the generator with a pretrained set

A. Joint Latent Codes and Generator Optimization

For a single image reconstruction, deep image prior solve

arg min \( \gamma \) \( \| y - AG\gamma(z) \|^2 \)

In this optimization, \( z \) is initialized randomly and kept unaltered.

To jointly optimize the latent codes and generator parameters for a video sequence, we use the similar formulation as in \[2\]

but optimize it over the \( z_t \) and \( \gamma \). The resulting optimization problem can be written as

\[
\text{minimize} \sum_{t=1}^{T} \| y_t - A_tG\gamma(z_t) \|^2. \quad (3)
\]

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 \( \hat{x}_t = G\gamma(\hat{z}_t) \).

We initialize latent codes with samples drawn from a Gaussian distribution and normalize them to have unit norm. We initialize \( \gamma \) with random weights using the initialization scheme in \[24\]. Intilizing the generator with a pretrained set of weights can potentially serve as a good initialization and lead to good and faster convergence. We tested both variants, but observed 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 learned 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 \( \hat{z}_t \).

B. 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

\[
Z = [z_1, z_2, \ldots, z_T],
\]

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:

\[
\text{minimize} \sum_{t=1}^{T} \| y_t - A_tG\gamma(z_t) \|^2, \quad \text{s.t.} \quad \text{rank}(Z) = r. \quad (4)
\]

We solve \[4\] 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 vectors \( u_1, \ldots, u_r \) as

\[
z_t = \sum_{j=1}^{r} \alpha_{tj}u_j \quad (5)
\]

where \( \alpha_{tj} \) 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 \( r \left( \frac{1}{T} + \frac{1}{k} \right) \) 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.

III. EXPERIMENTAL RESULTS

A. Setup and Performance Comparison

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 \( 64 \times 64 \) and rotate by \( 2^\circ \) per frame for a total of 32 frames. We experiment on different real video sequences from publicly available KTH human action video dataset \[25\] and UCF101 dataset \[26\]. In Table \[I\] we report our results for ‘Handclapping’, ‘Handwaving’ and ‘Walking’ video
Algorithm 1: Generative Models for Low Rank Representation and Recovery of Videos

**Input:** Measurements $y_t$, measurement matrices $A_t$, A generator structure $G_r(\cdot)$

Initialize the latent codes $z_t$ and generator weights $\gamma$ 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. $\gamma$ via backpropagation.
- Update network weights $\gamma$.

**until** convergence or maximum epochs

**Output:** Latent codes: $z_1, \ldots, z_T$ and network weights: $\gamma$

sequences from KTH dataset; ‘Archery’, ‘Apply Eye Makeup’ and ‘Band Marching’ video sequences from UCF101 dataset. We centered and resized every frame in KTH videos to $64 \times 64$ and UCF101 videos to $256 \times 256$ pixels.

We used the well-known DCGAN architecture [27] for our generators, except that we do not use any batch-normalization layer. The latent code dimensions for grayscale $64 \times 64$, RGB $64 \times 64$ and RGB $256 \times 256$ video sequences are 64, 256 and 512 respectively. We use Adam optimizer for generator weights optimization and SGD for latent code optimization. Unless otherwise mentioned, we use rank=4 constraint as low rank constraint because we empirically found that we need a least rank=4 for a video sequence with 32 frames to get comparable performance.

We show comparison with classical total variation minimization based TVAL3D (3D extension of TVAL3 [28]) algorithm and state-of-the-art untrained generative prior based deep decoder [20] on denoising, inpainting, and compressive sensing tasks. We use two different deep decoder settings: underparameterized deep decoder (UP deep decoder) and overparameterized deep decoder (OP deep decoder). Although the authors suggested deep decoder to be UP, we report the results for OP deep decoder as well because it shows better performance and its hyperparameters are tuned by the authors of deep decoder.

Other then denoising and inpainting, we performed compressive random projection experiments where we used separable measurements, $Y = P^T X P$, where $X, Y$ are reshaped versions of $x, y$ as 2D matrices, $P$ is a random projection matrix.

We report the results for denoising experiment at 20 dB SNR noise, inpainting experiment for 80% missing pixels and compressive sensing experiments for 20% available measurements in Table I. From the results, we can observe that joint optimization with/without low-rank constraint outperform TVAL3D algorithm and UP deep decoder. It performs at par with OP deep decoder. In Figure 2, we show reconstruction performance for denoising, inpainting and compressive sensing at different measurement rate or noise level for ‘Handwaving’ video sequence. We can get the similar observation from these curves as well. We report some reconstruction results for ‘Handwaving’ sequence in Figure 2. From the reconstructions, we can say that joint optimization performs at par with the comparing algorithms. It especially performs well in reconstructing details from masked frames.

### TABLE I: Reconstruction performance measured in terms of PSNR for different compressive sensing problems. We show comparison with TVAL3D (3D extension of TVAL3 [28]) and deep decoder [20]. The results are average over five experiments with different random measurement matrices (or noise in the case of denoising).

<table>
<thead>
<tr>
<th>Video Sequence</th>
<th>Rotating MNIST</th>
<th>Handclapping</th>
<th>Handwaving</th>
<th>Walking</th>
<th>Apply Eye Makeup</th>
<th>Archery</th>
<th>Band Marching</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Denoising for additive Gaussian noise of 20dB SNR</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TVAL3D</td>
<td>35.8</td>
<td>32.2</td>
<td>30.4</td>
<td>30.5</td>
<td>34.5</td>
<td>31.5</td>
<td>30.6</td>
</tr>
<tr>
<td>UP deep decoder</td>
<td>28.9</td>
<td>28.4</td>
<td>25.6</td>
<td>28.3</td>
<td>28.1</td>
<td>29.6</td>
<td>28.1</td>
</tr>
<tr>
<td>OP deep decoder</td>
<td>36.6</td>
<td>31.1</td>
<td>30</td>
<td>31</td>
<td>34.4</td>
<td>33</td>
<td>31.6</td>
</tr>
<tr>
<td>Joint optimization</td>
<td>36.9</td>
<td>32.7</td>
<td>30.7</td>
<td>31.2</td>
<td>36.1</td>
<td>32.1</td>
<td>31.3</td>
</tr>
<tr>
<td>Joint opt+lowrank</td>
<td>36.8</td>
<td>32.3</td>
<td>30.8</td>
<td>30.7</td>
<td>36.4</td>
<td>32</td>
<td>31.7</td>
</tr>
<tr>
<td><strong>Inpainting with 80% pixels randomly missing</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TVAL3D</td>
<td>21.1</td>
<td>29.2</td>
<td>23.8</td>
<td>24.5</td>
<td>28.2</td>
<td>27.1</td>
<td>24.8</td>
</tr>
<tr>
<td>UP deep decoder</td>
<td>25.5</td>
<td>26.5</td>
<td>23.3</td>
<td>26.3</td>
<td>27.2</td>
<td>29</td>
<td>23.3</td>
</tr>
<tr>
<td>OP deep decoder</td>
<td>30.1</td>
<td>30.2</td>
<td>26.7</td>
<td>27.9</td>
<td>24.5</td>
<td>32.5</td>
<td>26.2</td>
</tr>
<tr>
<td>Joint optimization</td>
<td>29.3</td>
<td>34.9</td>
<td>28.1</td>
<td>28.9</td>
<td>35.8</td>
<td>32</td>
<td>26.8</td>
</tr>
<tr>
<td>Joint opt+lowrank</td>
<td>29.5</td>
<td>34.3</td>
<td>27.3</td>
<td>27.8</td>
<td>36.6</td>
<td>30.4</td>
<td>27.6</td>
</tr>
<tr>
<td><strong>Spatial compressive sensing with compression rate = 0.2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TVAL3D</td>
<td>29.8</td>
<td>32.1</td>
<td>28.9</td>
<td>28</td>
<td>33.9</td>
<td>28.4</td>
<td>27.8</td>
</tr>
<tr>
<td>UP deep decoder</td>
<td>30</td>
<td>27</td>
<td>24.9</td>
<td>26.7</td>
<td>26.2</td>
<td>27.6</td>
<td>22.5</td>
</tr>
<tr>
<td>OP deep decoder</td>
<td>35.2</td>
<td>32.9</td>
<td>30.6</td>
<td>29</td>
<td>33.1</td>
<td>31.2</td>
<td>27.4</td>
</tr>
<tr>
<td>Joint optimization</td>
<td>35.3</td>
<td>35.6</td>
<td>29.7</td>
<td>28.9</td>
<td>36</td>
<td>29.3</td>
<td>27.8</td>
</tr>
<tr>
<td>Joint opt+lowrank</td>
<td>35.4</td>
<td>34.7</td>
<td>29</td>
<td>29.1</td>
<td>35.9</td>
<td>28.8</td>
<td>29.1</td>
</tr>
</tbody>
</table>

### B. 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 calculate memory requirement for gradient descent using torchsummary package [29]. For a single $64 \times 64$ RGB image, memory requirement for UP deep decoder, OP deep decoder and joint optimization is 2.75 MB, 66.48 MB and 2.06 MB respectively. For a single $256 \times 256$ RGB image, memory requirement for UP deep decoder, OP deep decoder and joint optimization is 44.03 MB, 1239.75 MB and 10.88 MB respectively. For a RGB video sequence with 32 frames, UP deep decoder will require 11, 304 x 32 (361,728) parameters while OP deep decoder will have 397, 056 x 32 (~12.7M) parameters. On the other hand we need 4,852,736 and 6,988,544 network parameters to represent RGB $64 \times 64$ and $256 \times 256$ video sequences, respectively in joint optimization method with DCGAN generator.

Because of the huge memory requirement, OP deep decoder is not suitable for optimization over entire video sequence whereas the low capacity hinders UP deep decoder from generating entire video sequence.

### C. Similarity Structure in the Latent Codes

To investigate the similarity structure in the latent codes obtained by joint optimization, we performed another experiment in which we concatenated 16 frames from each of the six different video sequences (‘Handwaving’, ‘Handclapping’,
Fig. 2: (a)-(c) are some reconstructions for Handwaving sequence for (a) denoising with 20 dB SNR noise level, (b) inpainting from masked frames with 80% missing pixels, (c) compressive sensing from 20% measurements. The deep decoder reconstruction here correspond to OP deep decoder structure because quantitative reconstruction results for UP deep decoder are significantly worse. Joint optimization with/without low rank constraint performs at par with other comparing algorithms. (d)-(f) are reconstruction quality curves for (d) denoising (e) inpainting (f) compressive sensing for Handwaving sequence. We can observe that joint optimization performs consistently for different measurement rates or noise level.

Fig. 3: 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 the frames of same video sequences found by joint optimization.

‘Walking’, ‘Archery’, ‘Apply Eye Makeup’, and ‘Band Marching’, in the same order) to create a new sequence with 96 frames. We performed compressive sensing experiment on this video sequence with 20% measurements.

Fig. 4: Linear representation of latent space of Rotating MNIST sequence using rank=2 constraint for image inpainting task with 80% missing pixels. The annotations for each point denotes the latent code corresponding to $t^{th}$ frame.

We mentioned that using low rank constraint we can represent the video sequences in much lower dimensional space. If the generator function is continuous it makes sense that the latent space representation of a video sequence will retain their sequential structure in some low dimensional representation. We demonstrate one such example using rank=2 constraint on the latent codes while reconstructing ‘Rotating MNIST’ sequence from its masked version with 80% pixels missing. As we are enforcing, rank=2 constraint taking mean and first principal component, the latent codes should fall on line. In Figure 4, we represent the latent codes in a 2D plane using 2 orthogonal basis vectors. $t^{th}$ point in Figure 4 represent latent code of $t^{th}$ frame. We can observe that latent codes are maintaining sequence in their 2D dimensional representation. For complex motions, it might take higher dimensional representation to observe such sequential pattern.
REFERENCES


