Video denoising using low rank matrix completion

Kalp Vyas, Sahil Garg

May 2, 2022

CS754 : Advanced Image Processing

Abstract

Most existing video denoising algorithms assume a single statistical model of image noise, e.g. additive Gaussian white noise, which often is violated in practice. In this project, we present a new patch-based video denoising algorithm capable of removing serious mixed noise from the video data. By grouping similar patches in both spatial and temporal domain, we formulate the problem of removing mixed noise as a low-rank matrix completion problem, which leads to a denoising scheme without strong assumptions on the statistical properties of noise.

1 Introduction

In this project, we have implemented the method of video denoising as shown in this paper. We have also compared the results of our algorithm with the one using the VBM3D (Video Block Matching and 3D filtering) library as well change in the reconstruction error with change in noise.

2 Problem to be solved

Let F $= \{f_k\}_{k=1}^{K}$ be an image sequence with K frames. Each image f_k is a sum of its underlying clean image g_k and the noise n_k:
$f_k = g_k + n_k$ (1)
The goal of video denoising is to recover $G = \{g_k\}_{k=1}^{M}$ by removing n_k from f_k.

To exploit the temporal redundancy in a video, we take a patch-based approach to jointly remove image noise n_k for all image frames. For each image f_k, consider one image patch $p_{j,k}$ of size $n \times n$ (say n = 8) centered at pixel j. We set this patch as a reference patch and search for patches that are similar to $p_{j,k}$ in all other images and within the neighborhood of the image f_k itself. Let's say m patches $\{p_{i,j,k}\}_{i=1}^{M}$ similar to $p_{j,k}$ are found in both spatial and temporal domain. If we represent each patch $p_{i,j,k}$ as a vector $p_{i,j,k} \in \mathcal{R}^{n^2}$ by concatenating all columns of the patch, we define a $n^2 \times m$ matrix $P_{j,k}$ as follows:
$P_{j,k} = (p_{1,j,k}, p_{2,j,k}, ..., p_{m,j,k})$ (2)
Then, we can re-write (1) in the form of patch matrix:
$P_{j,k} = Q_{j,k} + N_{j,k}$ (3)
where $Q_{j,k}$ denotes the patch matrix from the clean image g_k and $N_{j,k}$ denotes the noise.

If the data is free of noise and patch matching is also perfect, all column vectors in $Q_{j,k}$ have similar underlying image structures, the rank of $Q_{j,k}$ should be low and the variance of each row vector in $P_{j,k}$ should be very small. Hence, the problem becomes a low rank matrix recovery problem and the our approach to solving it is explained below.

3 The Solution

The solution is divided into 2 stages:

1. The reliable elements in $P_{j,k}$ are identified based on their deviation to the mean of all elements in same row. Let Ω denote the index set of all such elements.

2. Recover $Q_{j,k}$ from the incomplete version of $P_{j,k}$, denoted by $P_{j,k|\Omega}$, which is actually a matrix completion problem. That is, how to recover $Q_{j,k}$ from its noisy and incomplete observation $P_{j,k|\Omega}$ under the constraint that the rank of $Q_{j,k}$ is small. We consider the approach to estimate $Q_{j,k}$ from $P_{j,k|\Omega}$ by solving the following minimization problem :

$$min_Q ||Q||_*$$
$$s.t. ||Q|_\Omega - P|_\Omega||_F^2 \leq \#(\Omega)\hat{\sigma}^2$$

where $||.||_*$ is the nuclear norm (sum of the singular values of the matrix), $\#(\Omega)$ is the size of the set Ω, and $\hat{\sigma}$ is the estimate of standard deviation of noise from the noisy observations in Ω.

3.1 Stage 1: RAMF and Patch matching

We don't apply patch matching algorithm directly on the raw video data. Instead, a basic (intermediate) estimate of the video data is first obtained by using either some existing denoising technique or using the proposed algorithm, then the patch matching is done by using the intermediately denoised video data, which improves the accuracy of patch matching compared to that using raw un-denoised data. We use an adaptive median filter for this purpose. The adaptive median filter was implemented as per this paper. The RAMF (Ranked order based Adaptive Median Filter) was used for this purpose.

RAMF: A 2 level adaptive filter implementation where the first level tests for the presence of residual impluses in the median filter output and the second level checks whether the center pixel itself has been corrupted. If not, the output of RAMF is replaced by the median filter output at the first level. On the other hand, if the first level asserts there is an impulse in the median filter output, then we simply increase the window size for the median filter and repeat the first-level test. For our case, we have used the maximum window size as 10. The code for this has been implemented in */Codes/adaptive_median_filter.m* .

Patch matching: We have simply used MAD (Mean Absolute Difference) to compare the match between any 2 patches. Reference paper for using this technique is this. We have used 50 frames and 8 x 8 patches as suggested by the paper and sampled them by intervals of 4 x 4 which means we have overlapping patches. Further, for each patch, 5 most similar patches are used in each image frame. Finding patches was done using brute force due to the small amount of patches required. The code for this was implemented in */Codes/patchmatcher.m*

3.2 Stage 2: Matrix recovery algorithm

1) VBM3D: For this algorithm, we directly used their public github repo for the code. The repo used is here. For this method, we tried running the algorithm without any RAMF filter and then ran it after running the RAMF filter and observed major differences in the reconstruction rate, which was significantly higher when the adaptive median filter was used. No patchmatching was required for this algorithm. The code for this is in */Codes/BM3D*

2) Fixed Point Iteration: This was the main algorithm mentioned in the paper.
For each patch, similar patches are found in both spatial and temporal domain by using the patch matching algorithm described in the previous section to form the matrix $P_{j,k}$. The set of missing elements of $P_{j,k}$ have two subsets: the first subset are those pixels corrupted by impulsive noise using the adaptive median filter based impulsive noise detector. The second subset includes the pixels whose value differs from the mean of the corresponding row vector by the amount larger than a pre-defined threshold. Then Ω is formed by including the index of all remained pixels. We have taken the pre-defined threshold as 2 times the standard deviation of the row being considered.

The algorithm used is:

1. Set $Q^{(0)} := 0$.
2. Iterate till $||Q^{(k)} - Q^{(k-1)}||_F \leq \epsilon$,

   $$R^{(k)} = Q^{(k)} - \tau P_\Omega(Q^{(k)} - P),$$
   $$Q^{(k+1)} = D_{\tau\mu}(R^{(k)}),$$

   where $\mu = (\sqrt{n_1} + \sqrt{n_2})\sqrt{p}\hat{\sigma}$ and $1 \leq \tau \leq 2$ are pre defined parameters, $n_1 \times n_2$ is the size of the patch matrix, p is the ratio of no of pixels in Ω over total number of pixels, $\hat{\sigma}$ is obtained by calculation average of the variances of all elements $\in \Omega$ in each row, D is the soft shrinkage operator and P_Ω is the projection operator.

   $P_\Omega(Q)(i,j) = Q(i,j)$ if $(i,j) \in \Omega$ and 0 otherwise.
   $D_\tau(X) = U\Sigma_\tau V^T$, where $\Sigma =$ diag(max($\sigma_i - \tau$,0)), σ_i denotes the i^{th} largest singular value of X and $X = U\Sigma_T$ is the SVD for X.
3. Output $Q := Q^{(k)}$

We took $\epsilon = 0.00001$, maximum iterations as 30, $\tau = 1.2$, $n_1 = 64(8 \times 8), n_2 = 5$*no of frames $= 250$ since number of frames we are taking as 50. Also, all the tests were done on the 20^{th} frame of the video.

The code for the fixed point iteration algorithm has been implemented in */Codes/fixed_point_iteration.m*

The code for finding the missing values has been implemented in the test script which is in /Codes/test.mlx

4 Results

Following are the results obtained on running the code on coloured images with noise distribution as (σ, κ, s) where σ^2I refers to variance of the gaussian white noise added(0 mean), Poisson noise is added with zero mean and variance κ*original noiseless image and impulsive noise is added as 0 or 255 with probability s and no change with probability 1-s (All pixel values are between 0 and 255). The following are results on images with (20,5,0.3) as the noise parameters. These results have been obtained by using the script from /Codes/test_coloured.mlx and the noise was added to the images using /Codes/noise.m

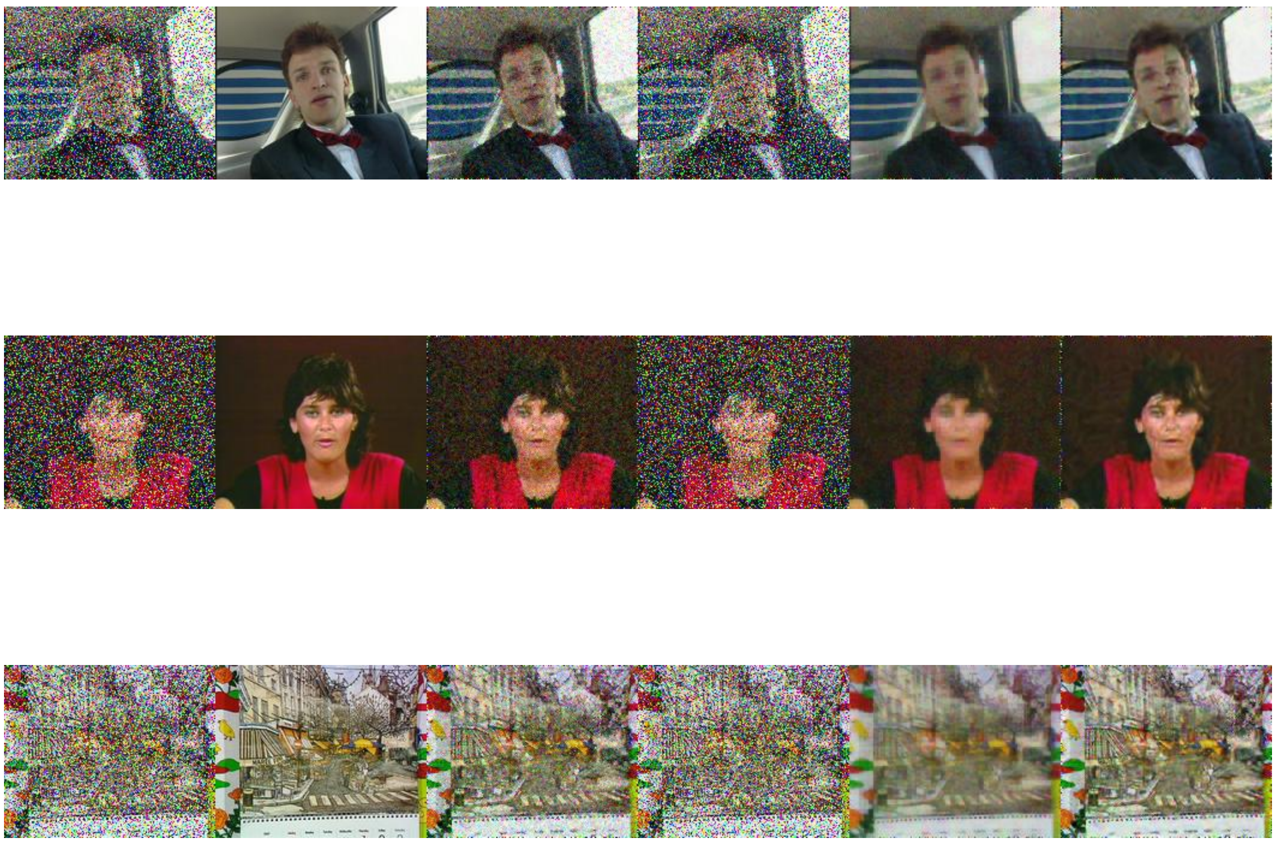

Figure 1: From left to right, a) image with 3 kinds of noise added; b) The original image; c)The noisy image after the adaptive median filter; d) Using VBM3D to denoise without using the median filter; e) Results of the low rank matrix completion based method (Fixed Point Iteration); f) Denoising by VBM3D with the median filter

Visual Observations:

1. As we can observe, the VBM3D method performs very poorly without the adaptive median filter and there is almost no denoising done but if it is applied after the adaptive filter, we get very good reconstruction.

2. Our method using fixed point iteration performs fairly well and ends up denoising all the noises pretty well with slight blurring as a kind of a downside.

3. Even though the image after VBM3D denoising (with median filter) looks good, there is still slight visible noise which shows us that the noise is still left behind in the image, which is not the case for our algorithm for which although the image seems slightly blurred, there is almost no visible noise.

5 Mathematical Results

The denoised images were compared using their PSNR (Peak Signal to Noise Ratio) values. For a reconstructed image f^r from it's original image f,

$PSNR(f^r) = 10log_{10}\frac{255^2}{||f-f^r||_2^2}$

The value of σ was fixed at 10 and impulse and poisson noise factors were changed to observe changes in the result as follows: (These results have been obtained by using the script from /Codes/test.mlx) (For the tables, x axis: poisson noise, y axix: impulse noise)

1) For fixed point iteration method

s/κ	1	5	10
0.1	25.8902	24.4692	22.9293
0.2	24.4136	23.8554	22.0155
0.3	22.8956	23.0079	21.5168
0.4	20.8260	21.9958	21.2704
0.5	18.5328	20.5377	19.9606

2) For VBM3D (without filter)

s/κ	1	5	10
0.1	24.2434	23.5200	22.7879
0.2	20.7797	20.3570	19.8365
0.3	18.6319	18.4886	18.2168
0.4	17.2552	17.0148	16.8536
0.5	15.7066	15.7737	15.7141

3) For VBM3D(with filter)

s/κ	1	5	10
0.1	25.9051	25.2723	24.7474
0.2	25.7057	25.0327	24.3249
0.3	25.4020	24.8152	24.0820
0.4	24.9798	24.2897	23.8507
0.5	24.4748	23.9188	23.2361

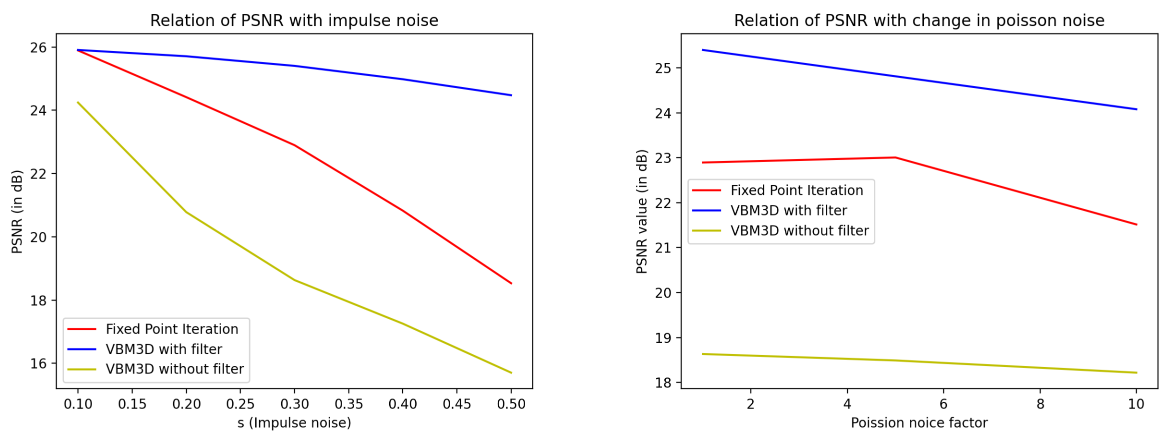

Figure 2: For impulse noise graph, $\kappa = 1$ and for the poisson noise graph, s=0.3

Observations

1. The VBM3D with filter seems to perform better than our algorithm which performs better than the VBM3D without filter.
2. The PSNR values for our algorithm and VBM3D without filter drop heavily with increase in impulse noise, whereas VBM3D with filter gives somewhat more stable results.
3. All the 3 methods give almost similar PSNR values on changing κ which means that the poisson noise isn't affecting the PSNR as heavily as impulsive affects. Further, as κ is increased for slightly bigger values of s (greater than 0.3), we can see that the fixed point iteration method has PSNR increasing with κ initially.
4. From the above 2 observations, we can conclude that the methods used are more sensitive to impulse noise rather than poisson and gaussian noise and a little change in impulse noise affects the result deeply.

6 Acknowledgement

We would like to thank the instructor of the course, Prof. Ajit Rajwade for giving us this opportunity to work on an interesting course project with practical applications.

External libraries used: From the internet,

1. **BM3D:** used to test VBM3D algorithm on the test videos and compare its results with out results.
2. **yuv4mpeg2mov:** used to convert the videos in the .y4m extension into a MATLAB-movie format which can be used.

Links to the datasets used:
All datasets were taken from `https://media.xiph.org/video/derf/`

GitHub repo for the project: `https://github.com/kalp121212/CS754_Project`