CS 754: Advanced Image Processing
Project Report

Krushnakant Bhattad	Devansh Jain
190100036	190100044

May 14, 2021

Project Title:

Implementing Low-rank matrix completion algorithm for Video denoising and comparing it with other denoising algorithms like PCA and VBM3D method.

Main Reference Paper:

"Robust video denoising using low rank matrix completion"
by Hui Ji, Chaoqiang Liu, Zuowei Shen, Yuhong Xu.
Link to the paper: `https://ieeexplore.ieee.org/document/5539849`

Data set used:

The original data set can be accessed at: `https://media.xiph.org/video/derf/`

Validation Strategy:

We compared our results to that from the methods with impulsive noise pre-processing, like PCA and VBM3D, with respect to their PSNR values (Peak Signal to Noise Ratio) and also visually.

Associated GitHub Repository:

The GitHub repository can accessed at: `https://github.com/devansh-dvj/Video-denoising/`

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. The paper presented a new patch-based video denoising algorithm capable of removing serious mixed noise from the video data.

The principle is to group similar patches in both spatial and temporal domain to 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. The resulting nuclear norm related minimization problem can be efficiently solved using various techniques, one of which was implemented by us - Singular Value Thresholding algorithm.

COMMON ABBREVIATIONS:

LRMC - Low Rank Matrix Completion
PCA - Principal Component Analysis
SVD - Singular Value Decomposition

The Library

Adaptive Median Filter (`adapmedfilt`)

It is step-by-step median filter based on Citation 16 of the Main reference Paper - "Adaptive median filters: new algorithms and results" by H. Hwang and R. A. Haddad.

We consider the first Noise model and use RAMF (ranked-order based adaptive median filter) with a bit of tweaking for computational efficiency.
The filter is based on two level - the first level tests for the presence of residual impulses in the median filter output, and the second level tests whether the center pixel itself is corrupted by an impulse or not.

The paper also shows that RAMF is superior to the nonlinear mean L_1 filter in removing positive and negative impulses while simultaneously preserving sharpness.

Code file: `libs/adapmedfilt/adapmedfilt.m`
Test file: `libs/adapmedfilt/adapmedfilt_test.m`

Noise Model (`noisemodel`)

In all our experiments, we have taken into account three types of noises - Gaussain, Poission and Impulsive.

Gaussian noise (amplifier noise) is zero mean with pixel independent variance $\sigma^2 I$.
Poisson noise (shot noise) is zero mean and variance κg_k, where g_k is the pixel intensity.
Impulsive noise by dead pixels, converter or transmission errors and etc with noise level s (in percentage).

Code file: `libs/noisemodel/noisemodel.m`
Test file: `libs/noisemodel/noisemodel_test.m`

Patch Matching (`patchmatcher`)

We use K image frames ($K = 50$ by default, but can be varied with experiments).
The patch size is set to 8×8.
We sample the reference patches per frame, with a sample interval of 4×4 pixels.
That means we'd have overlapping patches. We also set the range of image intensity to $[0, 255]$.

For each reference patch, 5 most similar patches are used in each image frame based on the ℓ_1 norm distance function.
We use brute force search to find these similar patches.
Thus, totally $5K$ patches are stacked for the reference patch, and thus the column dimension of the data matrix in the algorithms is $5K$.

Code file: `libs/patchmatcher/patchmatcher.m`
Test file: `libs/patchmatcher/patchmatcher_test.m`

PCA based Denoising (`pcai`)

Given the $5K$ patches $\{b_q\}_{q=1}^{5K}$, with similar underlying image structures, we now seek to remove their noise.

Let the size of each patch be $S \times S = D$.
We treat each patch $\{b_q\}$ as a D-dimensional vector.

Since all the matched patches have a similar underlying image structure, we assume that their noiseless patches lie in a low dimensional subspace, centered at u_0 and spanned by bases $\{u_d\}_{d=1}^{C}$.

Let b'_q be the denoised patch for b_q: $b'_q = u_0 + \sum_{d=1}^{C} u_d f_{q,d}$
where $f_{q,d}$ are the coefficients.

The cost function to be minimised is: $F(\{b'_q\}) = \sum_{q=1}^{5K} \left\| b_q - b'_q \right\|_\sigma^2$

where $\|x\|_\sigma^2 = \sum_{i=1}^{D} \frac{x_i^2}{\sigma_i^2}$ is an element-wise variance-normalized ℓ_2 norm that accounts for the intensity-dependent noise.

The answer to this minimization problem is given by PCA.

We first compute the mean patch u_0.
Then, we center the data by subtracting the mean from the input patches.
Afterward, we normalise each of the D variables by multiplying ith element by σ_i (which is the standard deviation at the ith patch pixel).

Then, we apply SVD over on the matrix to obtain the bases, and retain the data only along the C most "influential" dimensions.

We then, de-normalise this data, and add the mean patch to each of them to finally get the dimension-reduced data matrix.
We choose C by trial and error.

Code file: `libs/pcai/pcai.m`
Test file: `libs/pcai/pcai_test.m`

LRMC based Denoising (`svti`)

We form the matrix $P_{j,k}$ using the matched patches. The set of missing elements of $P_{j,k}$ has a subset of pixels corrupted by impulsive noise using the adaptive median filter based impulsive noise detector. Then, Ω is formed by including the index of all remained pixels.

We solve the problem: $\min_Q \left(\frac{1}{2}\|Q|_\Omega - P|_\Omega\|_F^2 + \mu\|Q\|_*\right)$

where $\|Q\|_*$ is the nuclear norm of Q, and μ is the lagrangian parameter, which we set according to $\mu = (\sqrt{n_1} + \sqrt{n_2})\sqrt{p}\hat{\sigma}$.

To solve this Low rank matrix completion(LRMC) problem, we use the Fixed point iteration algorithm (Singular Value Thresholding) as described below:

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

$$\begin{cases} R^{(k)} = Q^{(k)} - \tau\mathcal{P}_\Omega(Q^{(k)} - P), \\ Q^{k+1} = D_{\tau\mu}(R^{(k)}), \end{cases}$$

where μ and $1 \leq \tau \leq 2$ are pre-defined parameters, D is the shrinkage operator defined below and $\mathcal{P}_\Omega$ is the projection operator of Ω defined by

$$\mathcal{P}_\Omega(Q)(i,j) = \begin{cases} Q(i,j), & \text{if } (i,j) \in \Omega; \\ 0, & \text{otherwise.} \end{cases}$$

3. Output $Q := Q^{(k)}$.

Here, the shrinkage operator $D_r(X)$ is defined with respect to the Singular Value decomposition $X = U\Sigma V^T$ as $D_r(X) = U\Sigma_r V^T$ (from Candes' paper - Citation 7 of the main reference paper)
The main reference paper implemented another level of search for finding missing pixels using the mean and std. deviation of each row of the matrix. It can be used by setting `sec_missing` to true.

Code file: `libs/svti/svti.m`
Test file: `libs/svti/svti_test.m`

The Algorithms (`LRMC` and `PCA`)

Given an input noisy image sequence $\{\mathcal{I}\}_{i=1}^{K}$, we first apply the adaptive median filter to the images, so as to identify the pixels corrupted by impulsive noise and replace those damaged pixels by the median of small neighborhood.

Next, we create a patch-Array, where:
The patch size is set to 8×8, and
We sample the reference patches per frame, with a sample interval of 4×4 pixels.

De-noising a patch:

Given this patch-Array, we do Patch Matching to obtain a stack of $5K$ most similar patches.
Then, we apply the LRMC-based or PCA-based denoising on this stack of patches, to obtain denoised patch.

Doing this for each patch, we can effectively remove most noise from all patches.

The last step is to synthesize the denoised images from these denoised patches.

In our implementation, the image patches are sampled with overlapping regions. Thus, each pixel is covered by several denoised patches.

Then, the value of each pixel in images is determined by taking the average of denoised patches at this pixel.

This gives us the final denoised images.

LRMC Algorithm has four variants based on whether while iterating we update the patch array and whether we should consider second subset of missing pixels in `svti`

PCA Algorithm has two variants based on whether while iterating we update the patch array or not.

Code file for LRMC: `algos/LRMC/LRMC.m`
Code file for PCA: `algos/PCA/PCA.m`

The Results

Comparing Algorithms

Variants used:

- `LRMC1` : Low Rank Matrix Completion Algorithm with variant '01', $\tau = 1.5$, k_{max} 30, tol 1e-5
- `LRMC2` : Low Rank Matrix Completion Algorithm with variant '10', $\tau = 1.5$, k_{max} 30, tol 1e-5
- `PCA1` : Principal Component Analysis Algorithm with variant '0', C = 2
- `PCA2` : Principal Component Analysis Algorithm with variant '0', C = 4
- `PCA3` : Principal Component Analysis Algorithm with variant '0', C = 8
- `PCA4` : Principal Component Analysis Algorithm with variant '0', C = 16
- `VBM3D1` : VBM3D Algorithm with the adaptive median filter applied to the noisy video
- `VBM3D2` : VBM3D Algorithm without the adaptive median filter

$\{\sigma, \kappa\}$	s	PSNR of Reconstructed Frames							
		LRMC1	LRMC2	PCA1	PCA2	PCA3	PCA4	VBM3D1	VBM3D2
{10, 10}	10%	23.6945	23.5994	27.1342	27.3721	27.2584	26.4753	23.7962	17.5443
	20%	23.1713	23.1008	26.6927	26.8437	26.6462	25.8998	23.1268	15.5175
	30%	22.6781	22.5920	26.1415	26.2657	26.0366	25.2296	22.5524	14.1236
	40%	22.2009	22.1286	25.5636	25.5995	25.3423	24.4537	21.9991	13.0405
	50%	21.8527	21.7505	24.9397	24.9599	24.6227	23.6796	21.6151	12.2634
{50, 5}	10%	21.9801	22.0652	25.2479	25.2414	24.9745	24.2019	27.3927	23.9866
	30%	21.3173	21.3527	24.4847	24.4564	24.1792	23.3586	26.3149	17.0644
	50%	20.5023	20.4782	23.4038	23.2987	22.9143	21.9643	24.9581	14.3543

Comparing effects of change of parameters in LRMC

We work on $\sigma = 10$, $\kappa = 10$ and $s = 30\%$ for the following LRMC runs

tol	τ	k_{max}	PSNR of Reconstructed Frames			
			Variant '00'	Variant '01'	Variant '10'	Variant '11'
1e-5	1.0	30	21.6253	21.6466	21.4761	21.4979
		60	21.8325	21.8561	21.6951	21.7238
	1.5	30	21.8286	21.8527	21.6910	21.7201
		60	21.8327	21.8562	21.6952	21.7241
	2.0	30	21.8325	21.8560	21.6951	21.7239
		60	21.8327	21.8562	21.6952	21.7241