Patch Craft: Video Denoising by Deep Modeling and Patch Matching

Gregory Vaksman, ICCV , 2021

Saebom Lee

Abstract

With the emergence of CNN,
the patch-based framework has been abandoned.

We introduce a concept of **patch-craft frames**
-> to use the non-local self-similarity property of natural images

Our algorithm augments video sequences with patch-craft frames and feed them to a CNN

Contents

Sec1. Introduction

Introduction

Paragraph 01

Our goal in this paper to propose a novel video denoising strategy that builds on a synergy between the classics and deep neural networks.

⬇

A key feature we build upon is the evident spatio-temporal self-similarity existing in video sequences.

Paragraph 02

A spatial self-similarity property of natural image has been exploited extensively by classically oriented algorithms for solving image denoising problem.
These algorithms usually split the processed image into fully overlapping patches and arrange them into some structure according to their similarity.

Paragraph 03

the patch-based framework -> CNN

1. The reconstructed patches tend to be inconsistent on their overlaps.
2. the patch-based framework is the difficulty of combining it with CNNs.

Introduction

Paragraph 04

Several recent denoisers have combined the patch-based point of view within a deep-learning solution.

However, their performance is still challenged by leading convolutional networks, such as DnCNN and other networks

Paragraph 05

When turning to video processing, self similarity is further amplified due to the temporal redundancy.

⬇

Thus, harnessing non-local processing in video is expected to be even more effective and beneficial.

Paragraph 06

In contrast to the activity in image denoising, where many CNN-based schemes surpass classical algorithms, only a few video denoising networks have been shown to be competitive with classical methods.

Introduction

Paragraph 07

While CNN-based algorithms for video denoising, such as DVDnet and FastDVDnet, choose to work on whole frames rather than on patches, neural network based video denoising may still exploit self-similarity.

Paragraph 08

To leverage self-similarity while preserving the CNN's nature, We introduce the concept of patch-craft frames and use these as feature maps within the denoising process.

Paragraph 09

To process large amounts of data, we use a CNN composed of multidimensional separable convolutional (SepConv) layers.

Introduction

Paragraph 10

The processing pipeline of our proposed augmentation scheme for video denoising is composed of two stages.

1. We augment a noisy video sequence with patch-craft frames and feed the augmented sequence into a CNN built of SepConv layers.
2. We apply a temporal filtering, using a 3D extension of the DnCNN architecture.

Paragraph 11

We propose a neural network based video denoising scheme that consists of an augmentation of patch-craft frames, followed by a spatial and a temporal filtering.

The proposed method shows SOTA video denoising performance when compared to leading alternative algorithms.

Sec2. Patch-Craft Frames

Patch-Craft Frames

- We define the patch-craft frames as such auxiliary artificial frames built of patches taken from the current and surrounding frames.

- Constructing of the patch-craft frames:
  - Extract all possible overlapping patches of size √F x √F .
  - Find n nearest neighbors(most similar patches) for each extracted patch.
  - The n found neighbor patches are used for building the patch-craft frames where we utilize only their central parts of size √f x √f .

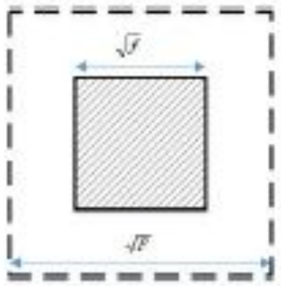


Figure 1: Patches of size $\sqrt{F} \times \sqrt{F}$ are used for nearest neighbor search. Their central $\sqrt{f} \times \sqrt{f}$ part are used for constructing patch-craft frames.

Patch-Craft Frames

- The patch-craft frames are created by stitching **non-overlapping** patches together.

  -> we build **f** groups of **n + 1** frames.

- Each group contains a copy of the processed frame and n patch-craft ones.
- The f groups differ by the patches' offsets.
- To avoid **block boundary artifacts**, we add **random noise** to a signal.
- We concatenate feature maps of scores with the f groups of n + 1 frames along the f dimension, passing the denoising network f + 1 groups of n + 1 frames for each processed original frame.

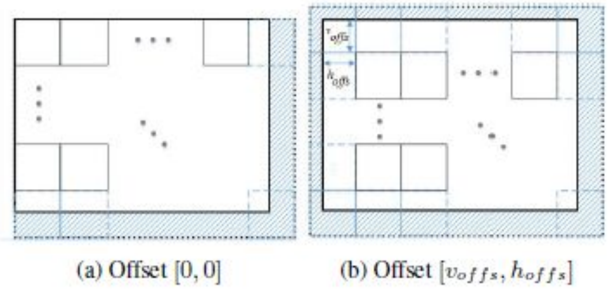
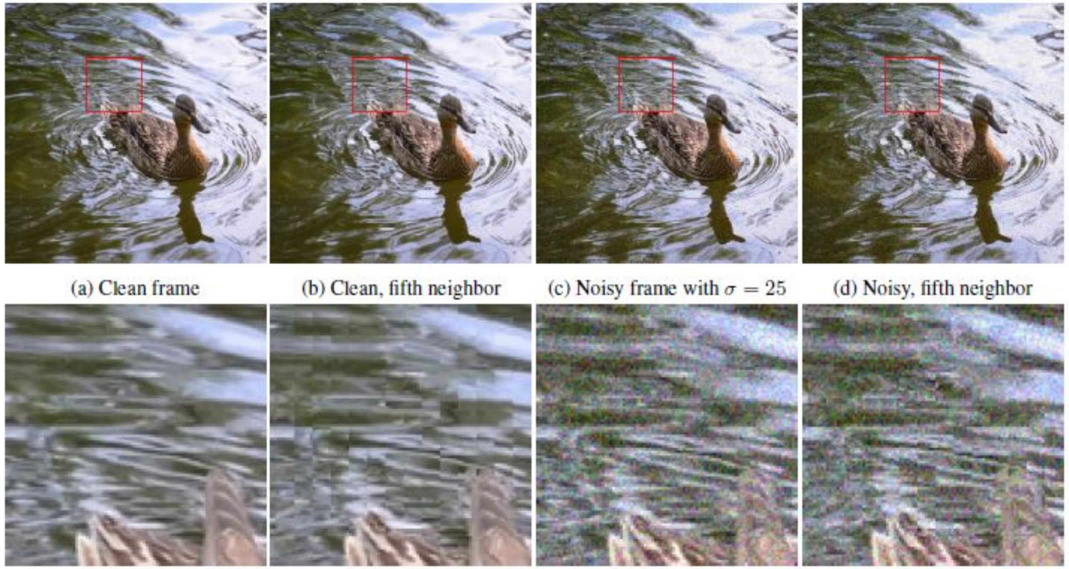


Figure 2: Splitting a frame to non-overlapping patches with different offsets. Figure 2a shows a splitting to patches without an offset ($[0, 0]$), while 2b shows a splitting to patches with an offset $[v_{offs}, h_{offs}]$. The white rectangle represents the processed frame, where the blue area represents a mirror reflection of the frame pixels.

Patch-Craft Frames


(a) Clean frame (b) Clean, fifth neighbor (c) Noisy frame with $\sigma = 25$ (d) Noisy, fifth neighbor

(e) Clean frame (f) Clean, fifth neighbor (g) Noisy frame with $\sigma = 25$ (h) Noisy, fifth neighbor

Sec3. The Proposed Algorithm

Sec 3-1. Separable Convolutional Neural Network
Sec 3-2. Temporal Filtering

Sec 3-1. Separable Convolutional Neural Network

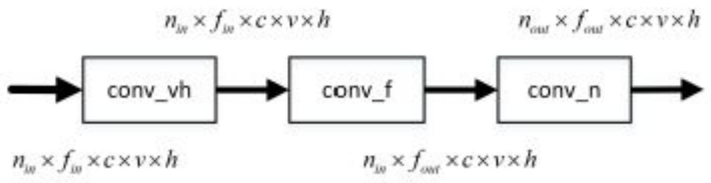

$n_{in} \times f_{in} \times c \times v \times h$
$n_{out} \times f_{out} \times c \times v \times h$
conv_vh
conv_f
conv_n
$n_{in} \times f_{in} \times c \times v \times h$
$n_{in} \times f_{out} \times c \times v \times h$

Figure 4: The SepConv layer.

*[v,h]: frame size
**c: number of color layers
***f: patch size
****n: number of neighbors to be used

- To process large amounts of data, we use separable convolutional layers.
- The SepConv layer is a separable convolutional layer composed of three convolutional filters, conv_vh, conv_ f, and conv_n.
- conv_vh -> mxm kernel : a local spatial prior, having n(in)f(in) groups of trainable convolution kernels.
- conv_f -> 1x1 kernel: a weighted patch averaging, having n(in) groups of trainable kernels.
- conv_n -> 1x1 kernel: a weighted neighbor averaging, having f(out)c groups of trainable kernels

Sec 3-1. Separable Convolutional Neural Network

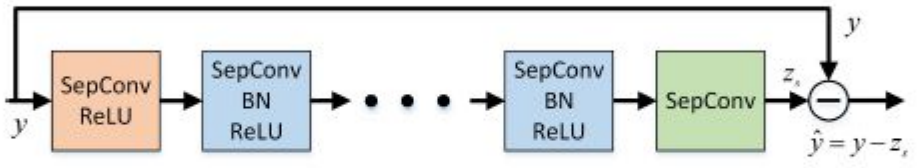


Figure 5: Our spatial denoising network S-CNN.

- Each SepConv layer reduces the number of neighbors by a factor of 2.
- The network operates in the residual domain predicting the noise z_s.
- The output frame $\hat{y}$ is obtained by subtracting the predicted noise from the corrupted frame y.

Sec 3-2. Temporal Filtering

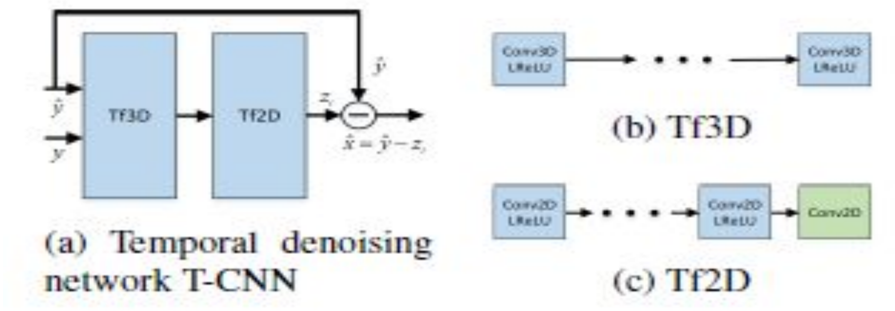


Figure 6: Our temporal filtering network.

- To guarantee **temporal continuity**, we apply temporal post-filtering, T-CNN, on the S-CNN output.

a. **T-CNN** : Similar to S-CNN, T-CNN works in the residual domain, predicting the noise z_t

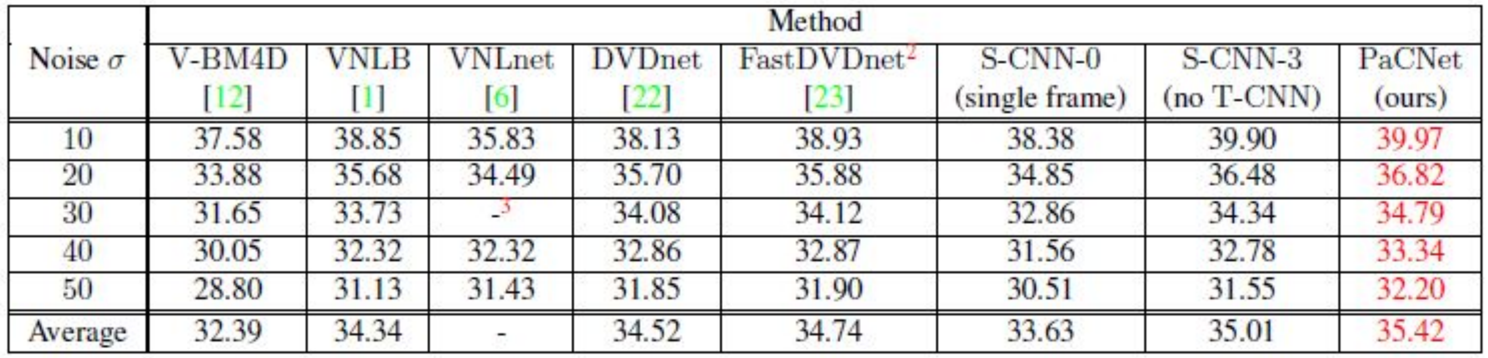
b. **Temporal Filter 3D(Tf3D)** is composed of Tt blocks consisting of a 3D convolutions with 3x3x3 kernels followed by Leaky ReLUs - each 3D kernel of Tf3D : no padding(temporal dimension), zero padding(spatially)

c. **Temporal Filter 2D(Tf2D)** consists of 2D convolutions with 3x3 kernels followed by Leaky ReLU - zero padding

Sec4. Experimental Results

Sec 4-1. Video Denoising
Sec 4-2. Single Image Denoising

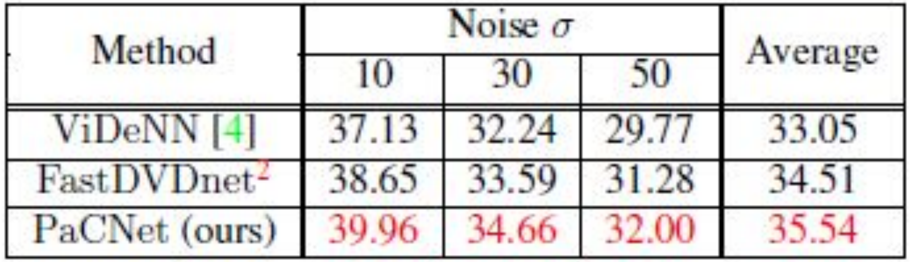
Sec 4-1. Video Denoising



Noise σ	Method							
	V-BM4D [12]	VNLB [1]	VNLnet [6]	DVDnet [22]	FastDVDnet[2] [23]	S-CNN-0 (single frame)	S-CNN-3 (no T-CNN)	PaCNet (ours)
10	37.58	38.85	35.83	38.13	38.93	38.38	39.90	39.97
20	33.88	35.68	34.49	35.70	35.88	34.85	36.48	36.82
30	31.65	33.73	-[3]	34.08	34.12	32.86	34.34	34.79
40	30.05	32.32	32.32	32.86	32.87	31.56	32.78	33.34
50	28.80	31.13	31.43	31.85	31.90	30.51	31.55	32.20
Average	32.39	34.34	-	34.52	34.74	33.63	35.01	35.42

Table 1: Video denoising performance on DAVIS [15] test set: Best PSNR is marked in red.

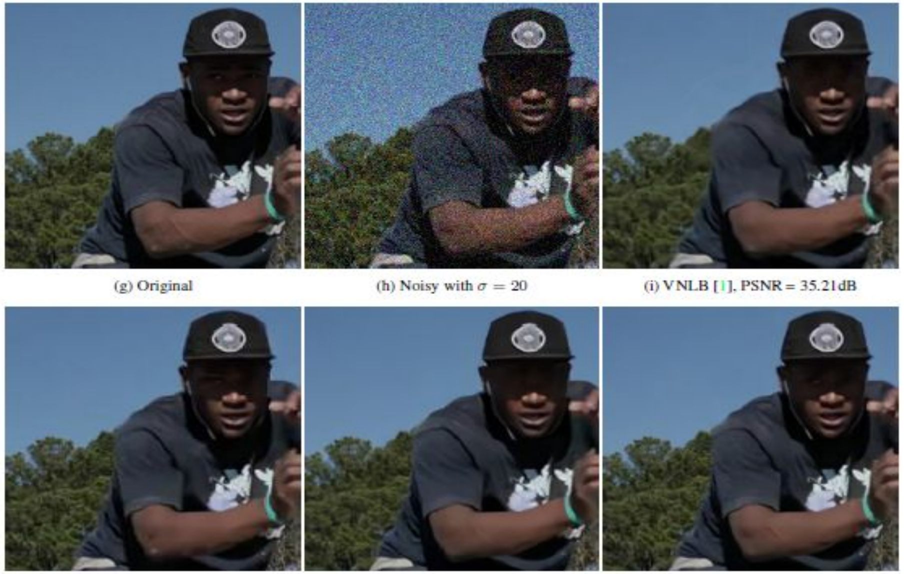
Sec 4-1. Video Denoising



Method	Noise σ			Average
	10	30	50	
ViDeNN [4]	37.13	32.24	29.77	33.05
FastDVDnet[2]	38.65	33.59	31.28	34.51
PaCNet (ours)	39.96	34.66	32.00	35.54

Table 2: Denoising for clipped Gaussian noise.

Sec 4-1. Video Denoising

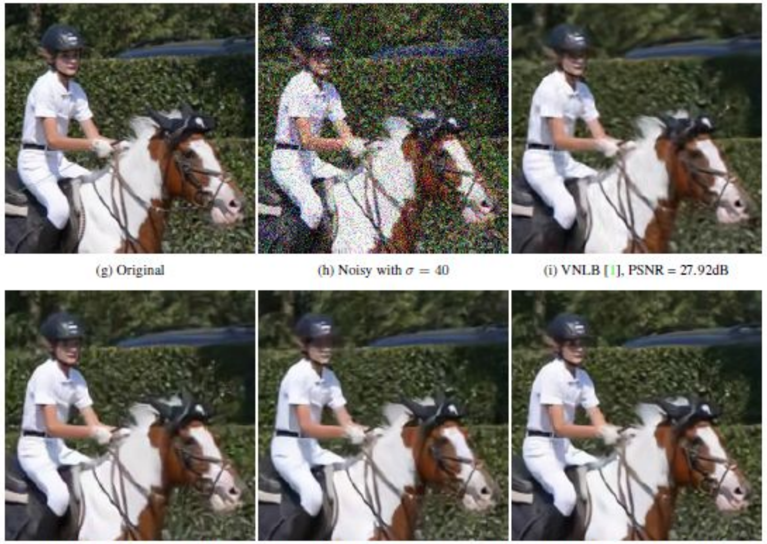


(g) Original (h) Noisy with $\sigma = 20$ (i) VNLB [1], PSNR = 35.21dB

(j) VNLnet [6], PSNR = 33.61dB (k) FastDVDnet [23], PSNR = 35.00dB (l) PaCNet (ours), PSNR = 36.11dB

Figure 7: Denoising example with $\sigma = 20$. The figure shows frame 61 of the sequence *skate-jump*. The PSNR values appearing in 7c, 7d, 7e and 7f refer to the whole frame, whereas those in 7i, 7j, 7k and 7l refer to the cropped area. As can be seen, PaCNet leads to better reconstructed result – see the eyes and the details in the background trees.

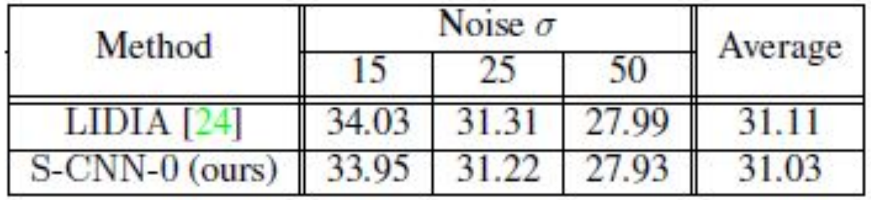
Sec 4-1. Video Denoising



(g) Original
(h) Noisy with $\sigma = 40$
(i) VNLB [1], PSNR = 27.92dB
(j) VNLnet [6], PSNR = 27.95dB
(k) FastDVDnet [23], PSNR = 28.23dB
(l) PaCNet (ours), PSNR = 29.07dB

Figure 8: Denoising example with $\sigma = 40$. The figure shows frame 48 of the sequence *horsejump-stick*. The PSNR values appearing in 8c, 8d, 8e and 8f refer to the whole frame, whereas those in 8i, 8j, 8k and 8l refer to the cropped area. As can be seen, PaCNet leads to better reconstructed results – see the face and the details in the background shrubs.

Sec 4-2. Single Image Denoising



Method	Noise σ			Average
	15	25	50	
LIDIA [24]	34.03	31.31	27.99	31.11
S-CNN-0 (ours)	33.95	31.22	27.93	31.03

Table 3: Single image denoising performance comparison.

Sec5. Conclusion

Conclusion

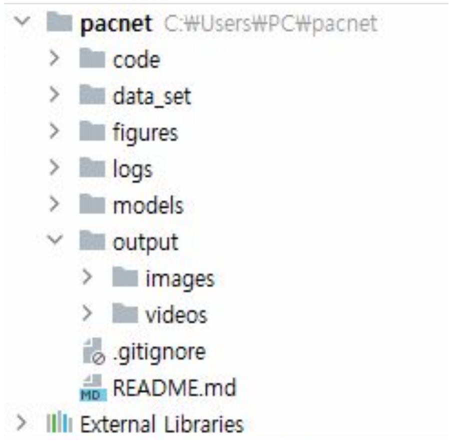
- Our method augments the processed video with patchcraft frames and applies spatial and temporal filtering on the enlarged sequence.
- The augmentation leverages non-local redundancy, similar to the way the patch based framework operates.
- The spatial denoising network consists of separable convolutional layers, which allow for reasonable memory and computational complexities.
- The temporal CNN reduces flickering by imposing temporal continuity.

Source code

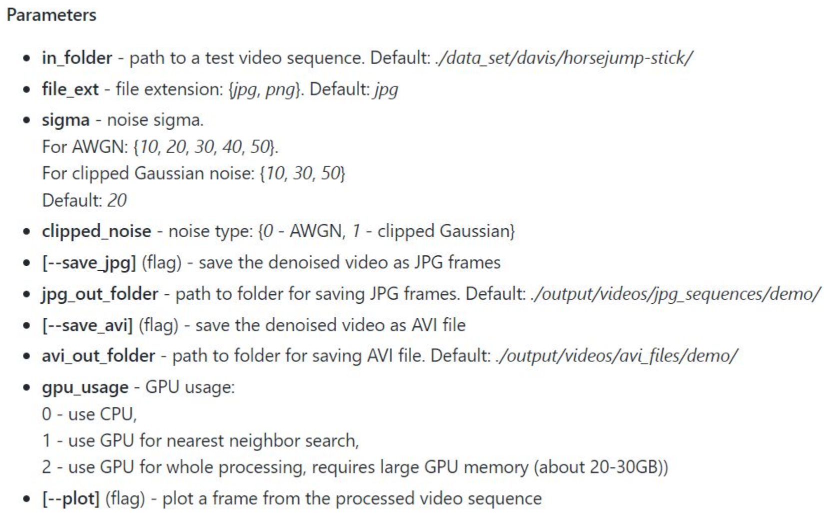
Source Code

- Framework : torch
- Code location : https://github.com/grishavak/PaCNet-denoiser
- Resources : Demo (pre-trained weights, class labels, test image)

Source Code



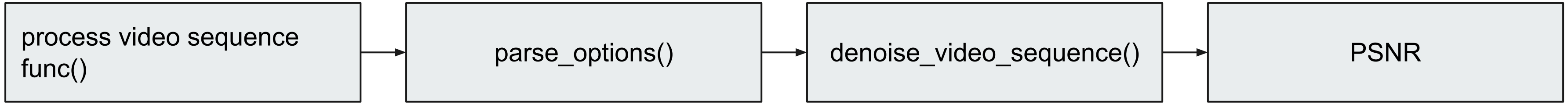
```
pacnet C:₩Users₩PC₩pacnet
  code
  data_set
  figures
  logs
  models
  output
    images
    videos
  .gitignore
  README.md
External Libraries
```



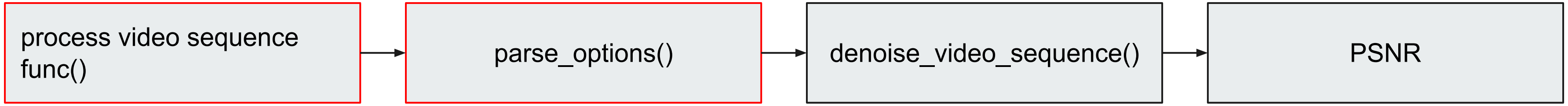
Parameters

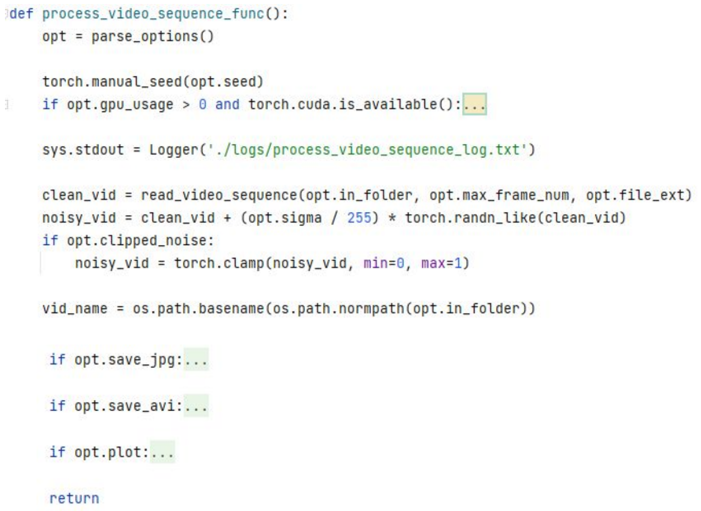
- **in_folder** - path to a test video sequence. Default: *./data_set/davis/horsejump-stick/*
- **file_ext** - file extension: {*jpg, png*}. Default: *jpg*
- **sigma** - noise sigma.
  For AWGN: {*10, 20, 30, 40, 50*}.
  For clipped Gaussian noise: {*10, 30, 50*}
  Default: *20*
- **clipped_noise** - noise type: {*0* - AWGN, *1* - clipped Gaussian}
- **[--save_jpg]** (flag) - save the denoised video as JPG frames
- **jpg_out_folder** - path to folder for saving JPG frames. Default: *./output/videos/jpg_sequences/demo/*
- **[--save_avi]** (flag) - save the denoised video as AVI file
- **avi_out_folder** - path to folder for saving AVI file. Default: *./output/videos/avi_files/demo/*
- **gpu_usage** - GPU usage:
  0 - use CPU,
  1 - use GPU for nearest neighbor search,
  2 - use GPU for whole processing, requires large GPU memory (about 20-30GB))
- **[--plot]** (flag) - plot a frame from the processed video sequence

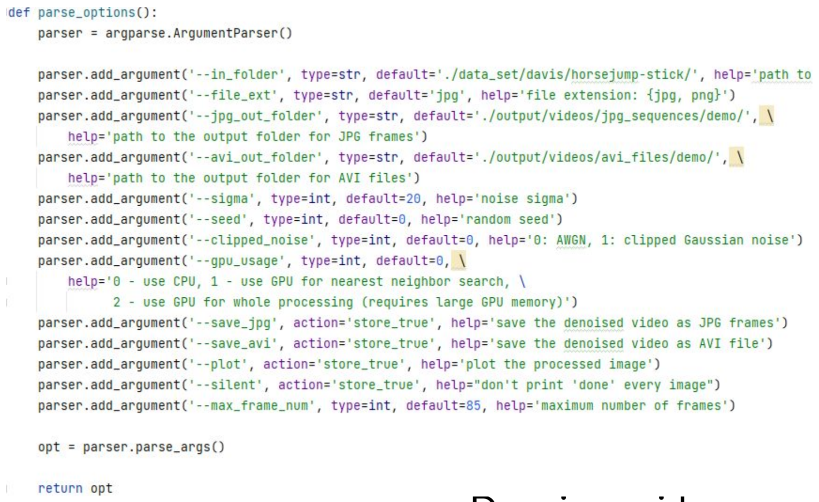
Source Code


process video sequence func()
parse_options()
denoise_video_sequence()
PSNR

Source Code

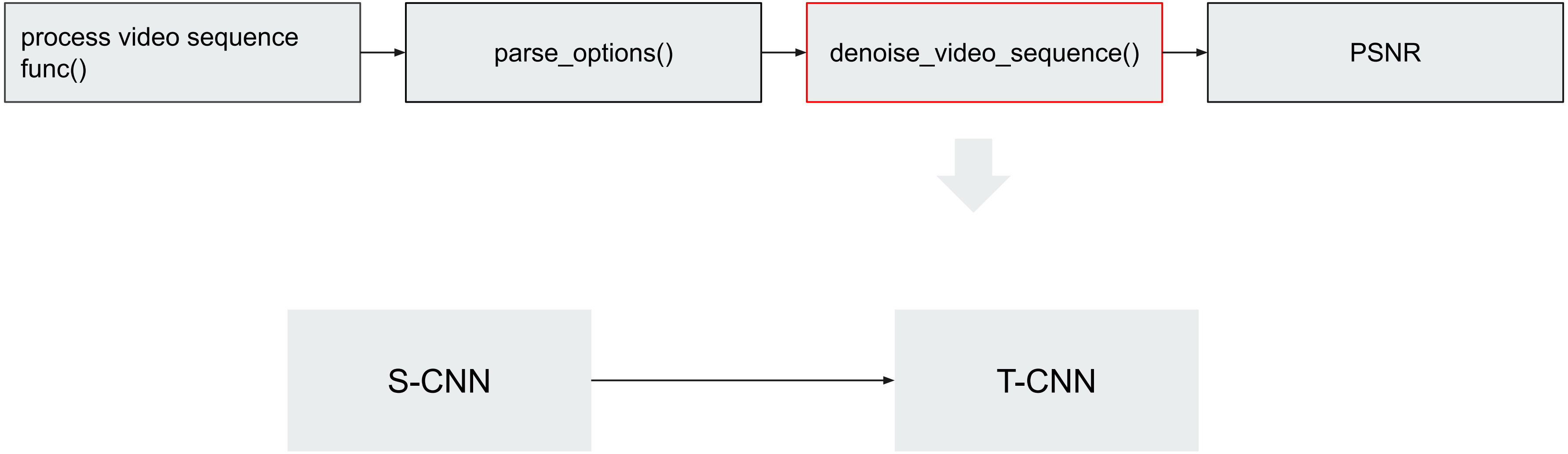

$$\begin{tikzcd}\boxed{\text{process video sequence func()}} \arrow[r] & \boxed{\text{parse\_options()}} \arrow[r] & \boxed{\text{denoise\_video\_sequence()}} \arrow[r] & \boxed{\text{PSNR}}\end{tikzcd}$$


```python
def process_video_sequence_func():
    opt = parse_options()

    torch.manual_seed(opt.seed)
    if opt.gpu_usage > 0 and torch.cuda.is_available():...

    sys.stdout = Logger('./logs/process_video_sequence_log.txt')

    clean_vid = read_video_sequence(opt.in_folder, opt.max_frame_num, opt.file_ext)
    noisy_vid = clean_vid + (opt.sigma / 255) * torch.randn_like(clean_vid)
    if opt.clipped_noise:
        noisy_vid = torch.clamp(noisy_vid, min=0, max=1)

    vid_name = os.path.basename(os.path.normpath(opt.in_folder))

    if opt.save_jpg:...

    if opt.save_avi:...

    if opt.plot:...

    return
```


```python
def parse_options():
    parser = argparse.ArgumentParser()

    parser.add_argument('--in_folder', type=str, default='./data_set/davis/horsejump-stick/', help='path to
    parser.add_argument('--file_ext', type=str, default='jpg', help='file extension: {jpg, png}')
    parser.add_argument('--jpg_out_folder', type=str, default='./output/videos/jpg_sequences/demo/', \
        help='path to the output folder for JPG frames')
    parser.add_argument('--avi_out_folder', type=str, default='./output/videos/avi_files/demo/', \
        help='path to the output folder for AVI files')
    parser.add_argument('--sigma', type=int, default=20, help='noise sigma')
    parser.add_argument('--seed', type=int, default=0, help='random seed')
    parser.add_argument('--clipped_noise', type=int, default=0, help='0: AWGN, 1: clipped Gaussian noise')
    parser.add_argument('--gpu_usage', type=int, default=0, \
        help='0 - use CPU, 1 - use GPU for nearest neighbor search, \
              2 - use GPU for whole processing (requires large GPU memory)')
    parser.add_argument('--save_jpg', action='store_true', help='save the denoised video as JPG frames')
    parser.add_argument('--save_avi', action='store_true', help='save the denoised video as AVI file')
    parser.add_argument('--plot', action='store_true', help='plot the processed image')
    parser.add_argument('--silent', action='store_true', help="don't print 'done' every image")
    parser.add_argument('--max_frame_num', type=int, default=85, help='maximum number of frames')

    opt = parser.parse_args()

    return opt
```

Denoise a video sequence

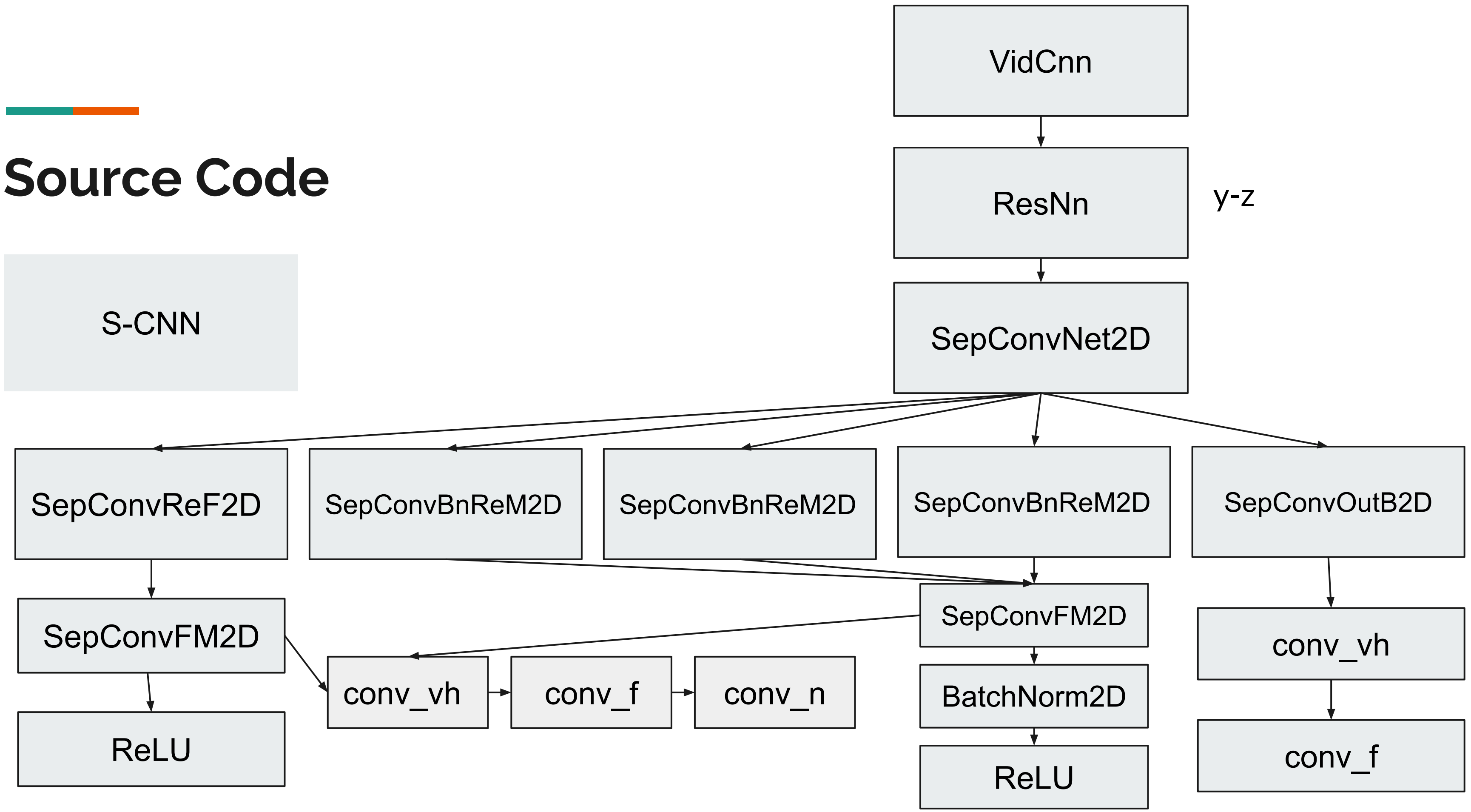
Source Code



Denoise a video sequence

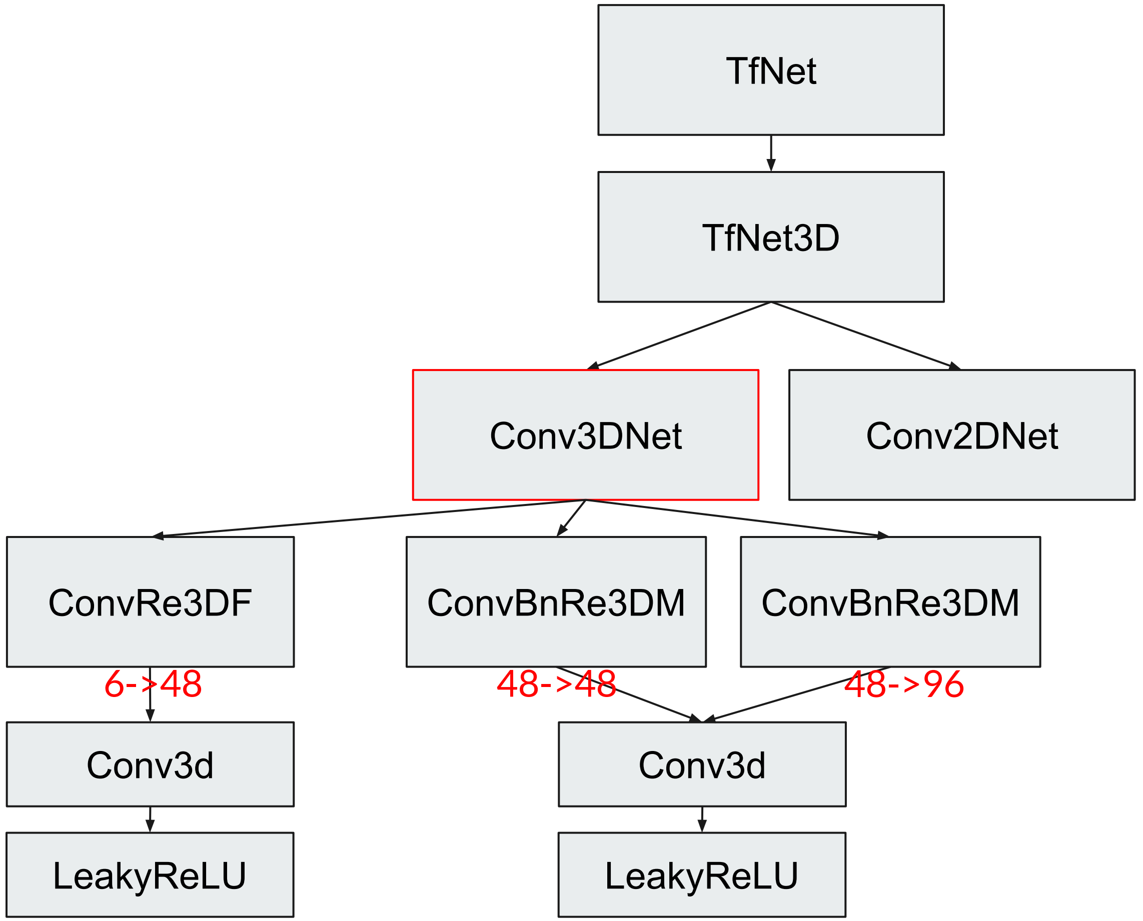
Source Code

S-CNN



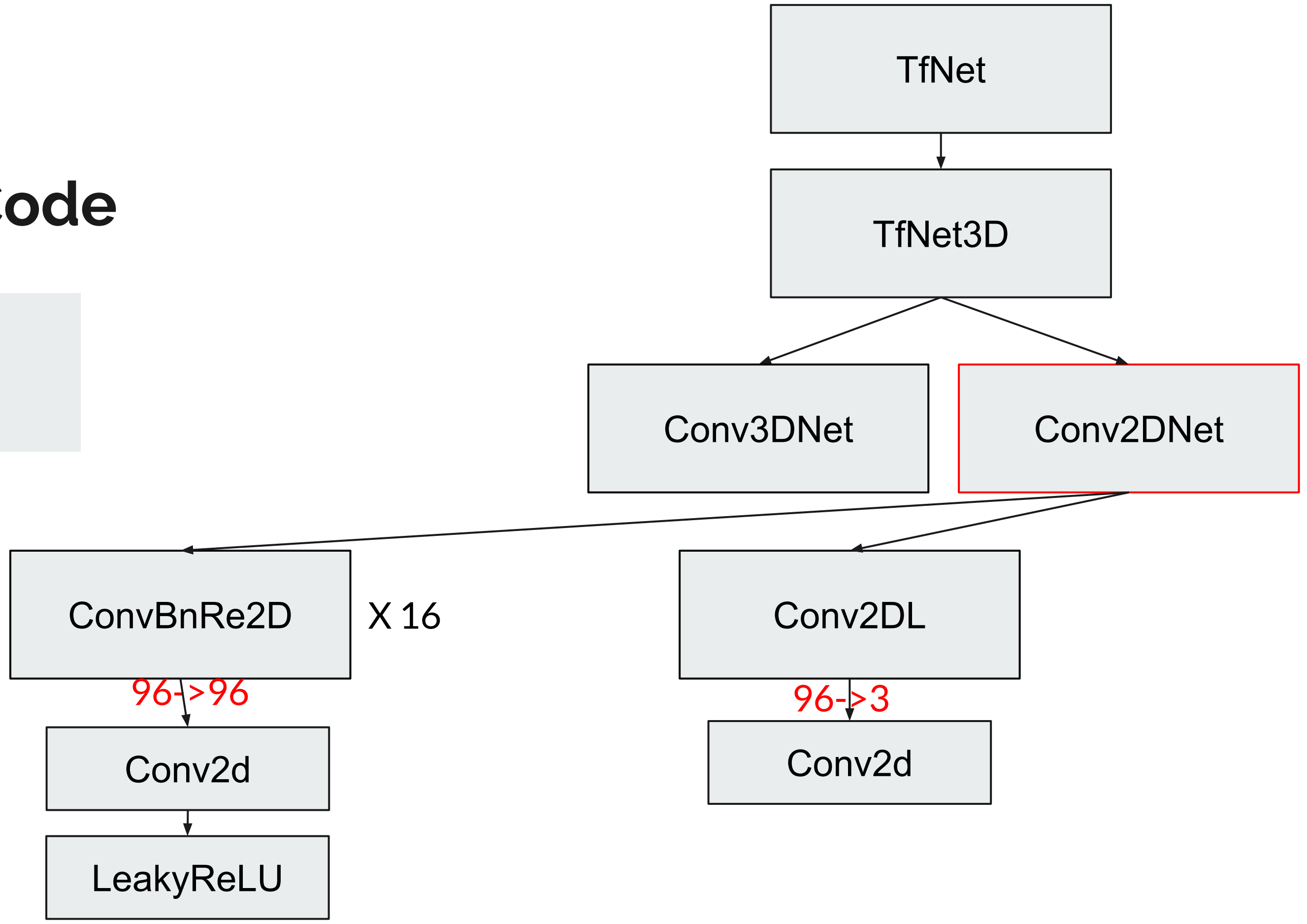
Source Code

T-CNN

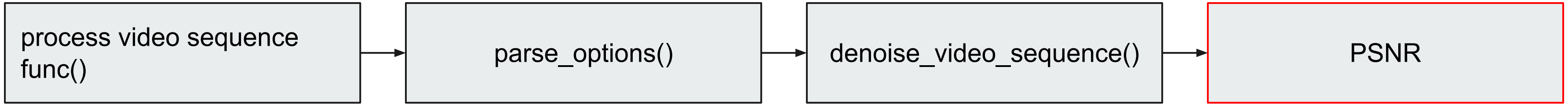


Source Code

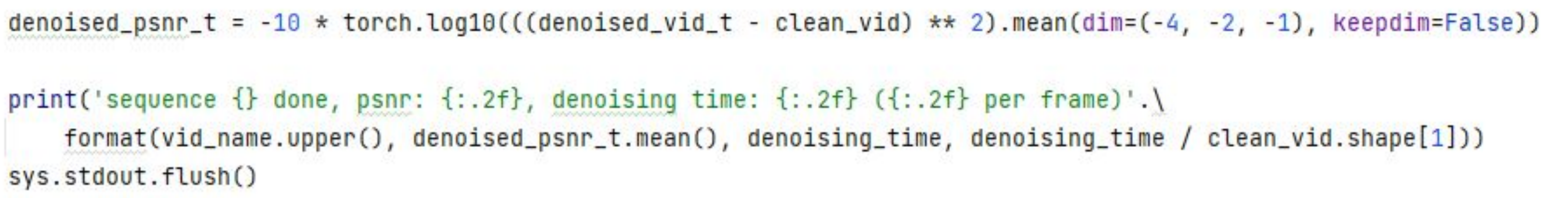
T-CNN



Source Code



$$\begin{aligned} PSNR &= 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) \\ &= 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \\ &= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE) \end{aligned}$$



```
denoised_psnr_t = -10 * torch.log10(((denoised_vid_t - clean_vid) ** 2).mean(dim=(-4, -2, -1), keepdim=False))

print('sequence {} done, psnr: {:.2f}, denoising time: {:.2f} ({:.2f} per frame)'.\
    format(vid_name.upper(), denoised_psnr_t.mean(), denoising_time, denoising_time / clean_vid.shape[1]))
sys.stdout.flush()
```

Denoise a video sequence

Source Code

./logs/process_video_sequence_log.txt

```
=> loading model state './models/s_cnn_video/model_state_sig20.pt'
S-CNN: frame 1/58 of sequence HORSEJUMP-STICK done, denoising time: 449.0940
S-CNN: frame 2/58 of sequence HORSEJUMP-STICK done, denoising time: 442.9916
S-CNN: frame 3/58 of sequence HORSEJUMP-STICK done, denoising time: 449.9192
S-CNN: frame 4/58 of sequence HORSEJUMP-STICK done, denoising time: 456.0987
S-CNN: frame 5/58 of sequence HORSEJUMP-STICK done, denoising time: 450.2463
....
S-CNN: frame 53/58 of sequence HORSEJUMP-STICK done, denoising time: 519.4554
S-CNN: frame 54/58 of sequence HORSEJUMP-STICK done, denoising time: 519.0117
S-CNN: frame 55/58 of sequence HORSEJUMP-STICK done, denoising time: 518.2064
S-CNN: frame 56/58 of sequence HORSEJUMP-STICK done, denoising time: 516.5954
S-CNN: frame 57/58 of sequence HORSEJUMP-STICK done, denoising time: 512.6775
S-CNN: frame 58/58 of sequence HORSEJUMP-STICK done, denoising time: 515.6536

=> loading model state './models/t_cnn/model_state_sig20.pt' T-CNN: sequence HORSEJUMP-STICK done, denoising time: 1.9373
sequence HORSEJUMP-STICK done, psnr: 34.21, denoising time: 29172.46 (502.97 per frame)
```

Source Code