Patch Craft: Video Denoising by Deep Modeling and Patch Matching

Gregory Vaksman, ICCV , 2021

Saebom Lee

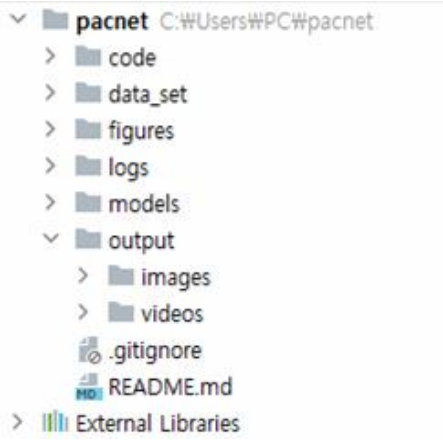
Source code

Source Code

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

- This code was tested with python 3.7, cuda 11.3 and pytorch 1.10.1 on NVDIA RTX3060 (12GB) (for video denoising with option gpu_usage=2).

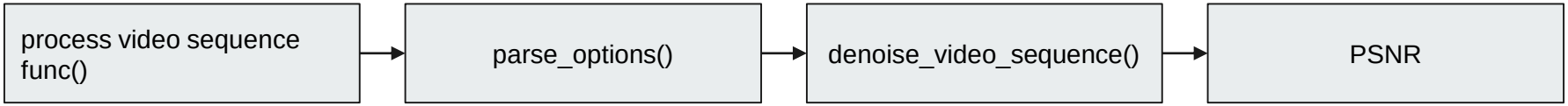
Source Code



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

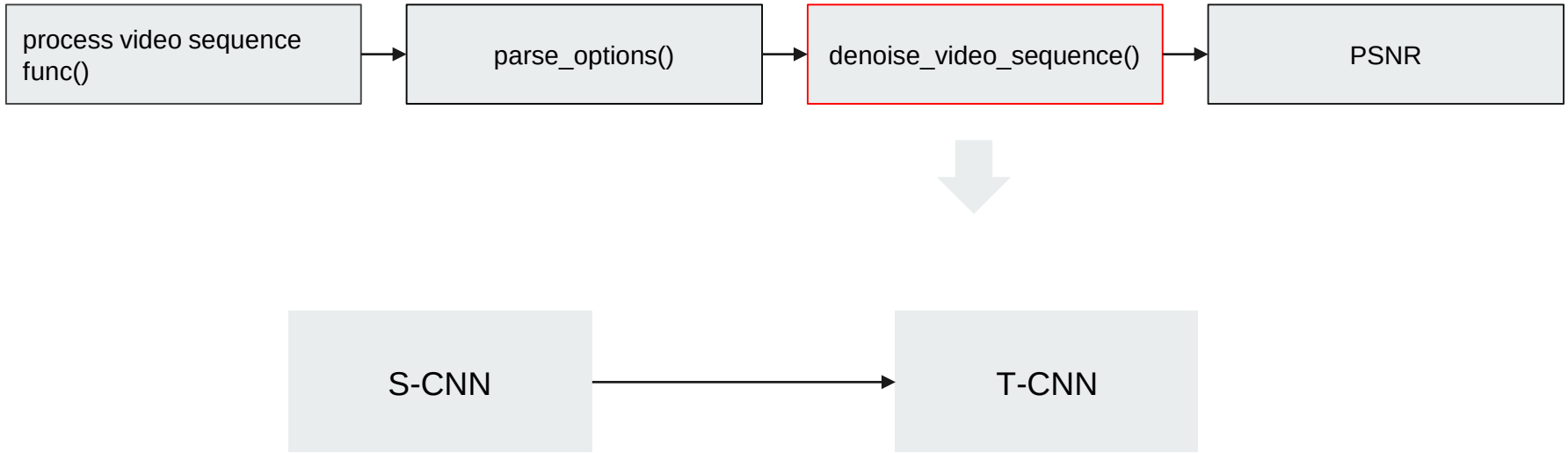
process video sequence func() → parse_options() → denoise_video_sequence() → PSNR

```
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
```

```
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 sequenc

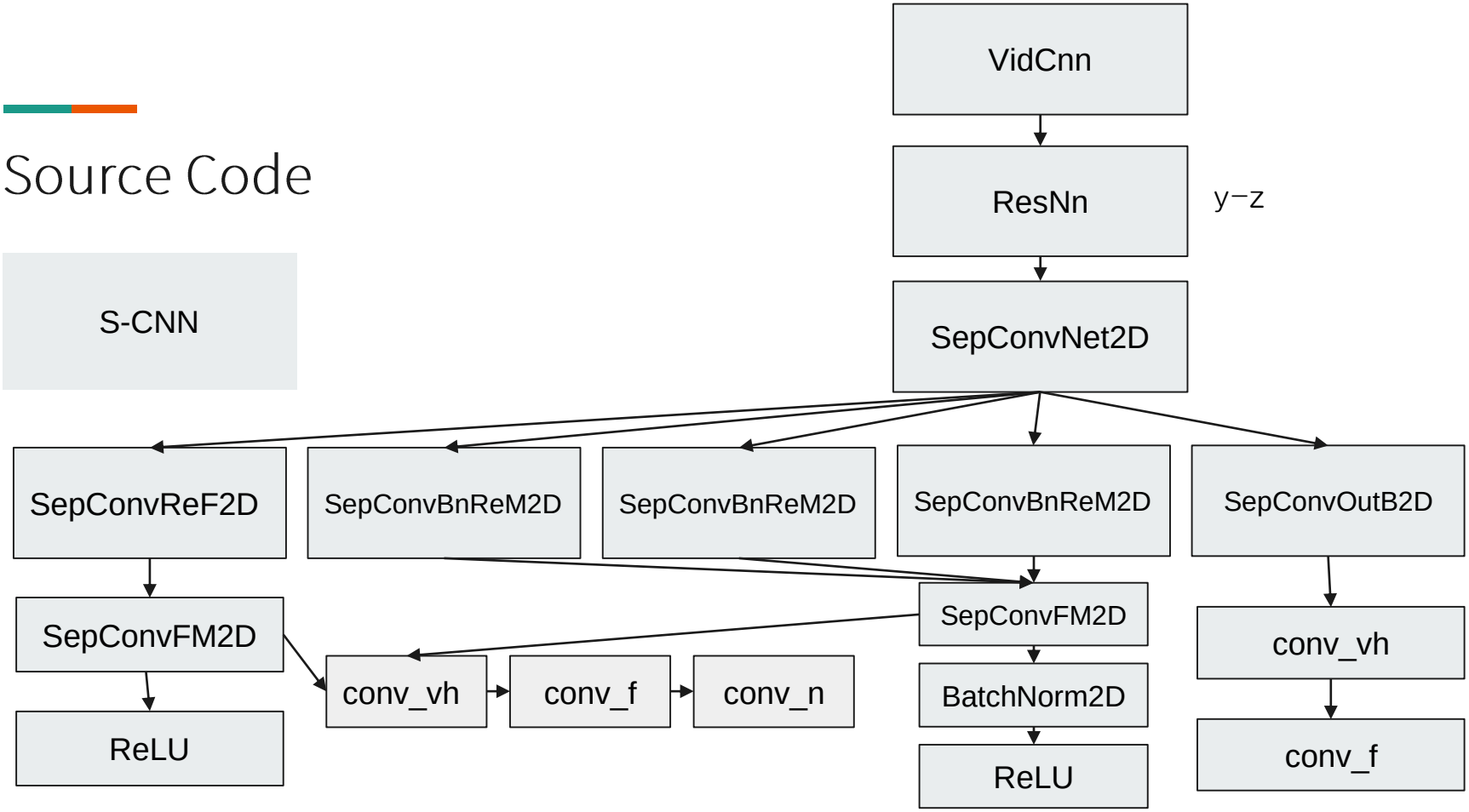
Source Code



Denoise a video sequenc

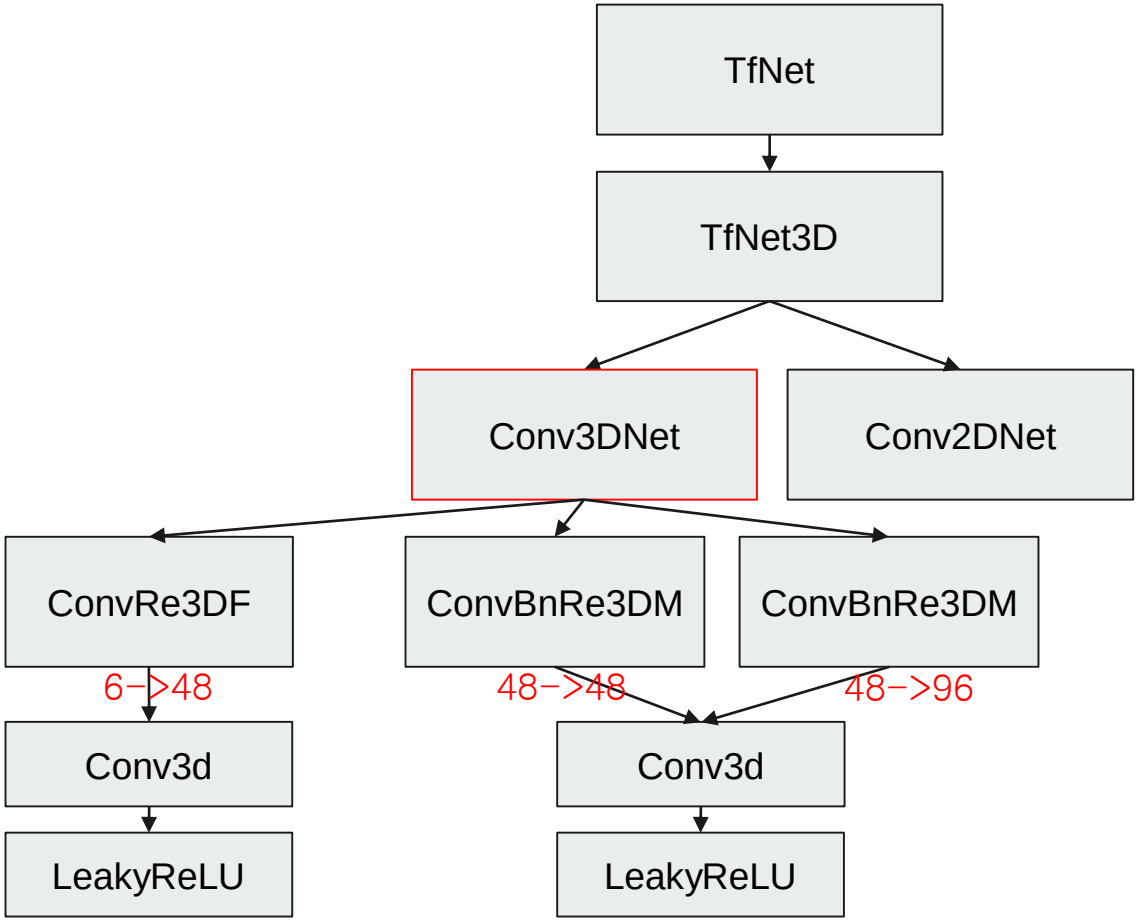
Source Code

S-CNN



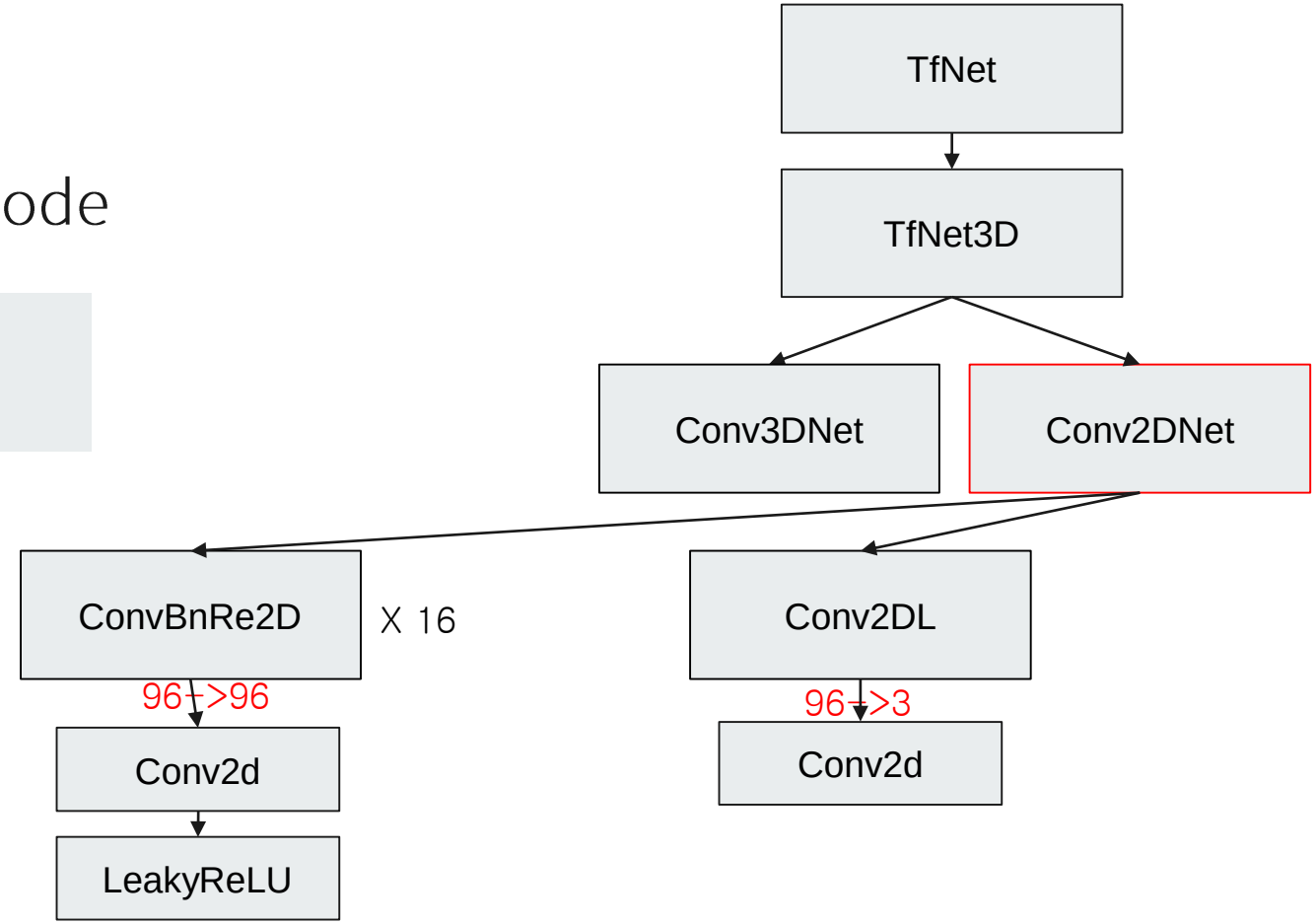
Source Code

T-CNN

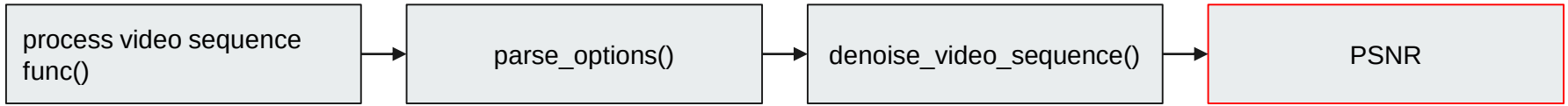


Source Code

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()
```

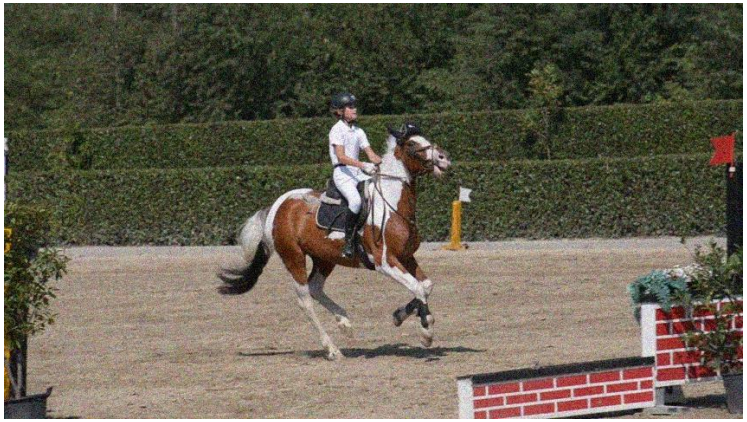
Denoise a video sequenc

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



Reviews

Reviews

<Advantages>

It is impressive that the model was constructed by combining only the advantages of the patch-based framework and the CNN method.

<Disdvantages>

The model structure is complicated that it takes a long time.

Thank you