Neural Sensors: Learning Pixel Exposures for HDR Imaging and Video Compressive Sensing with Programmable Sensors

–Supplemental Material–

Julien N. P. Martel, *Member, IEEE,* and L. K. Müller, and S. J. Carey, and P. Dudek, *Senior Member, IEEE*, and G. Wetzstein *Senior Member, IEEE*

1 Details about the Training of the Deep Network Architectures

1.1 Datasets

HDR : We use 297 high quality high-dynamic range images from HDRi Haven[1] downloaded at a resolution of 2048×1024px. The training set comprises 269 randomly sampled images (without replacement), and the testing set comprises the 28 remaining images.

For all the training experiments using 2D U-Nets, a training sample is produced as follows: an image is chosen at random from the 269 training images. First it is converted to grayscale. Then, a scale $s \sim \mathcal{U}([0.25, 0.40])$ is uniformly sampled, to rescale the high-quality high-resolution HDR image. Note that the minimum scaling factor of 0.25 is chosen so that 0.25×1024 is equal to the image sizes we consider of side length 256. Finally, a random crop of size 256×256 is performed on the rescaled image. The obtained 256×256 image is then randomly vertically and horizontally flipped, as well as rotated.

High-speed: We use the 100 video clips provided in the Need For Speed (NfS) dataset[2] in their 240FPS version. Those are converted to grayscale. The training set comprises 90 randomly sampled video clips (without replacement), and the testing set comprises the 10 remaining ones. Note, that in principle a low frame rate video dataset could also be used, as long as, the interframe discrepancy is not too high. A low-frame rate video with slow motion is seen the same by our method than a high-frame rate video with higher discrepancy since the networks do not have a real notion of time else than through the concatenation of successive frames.

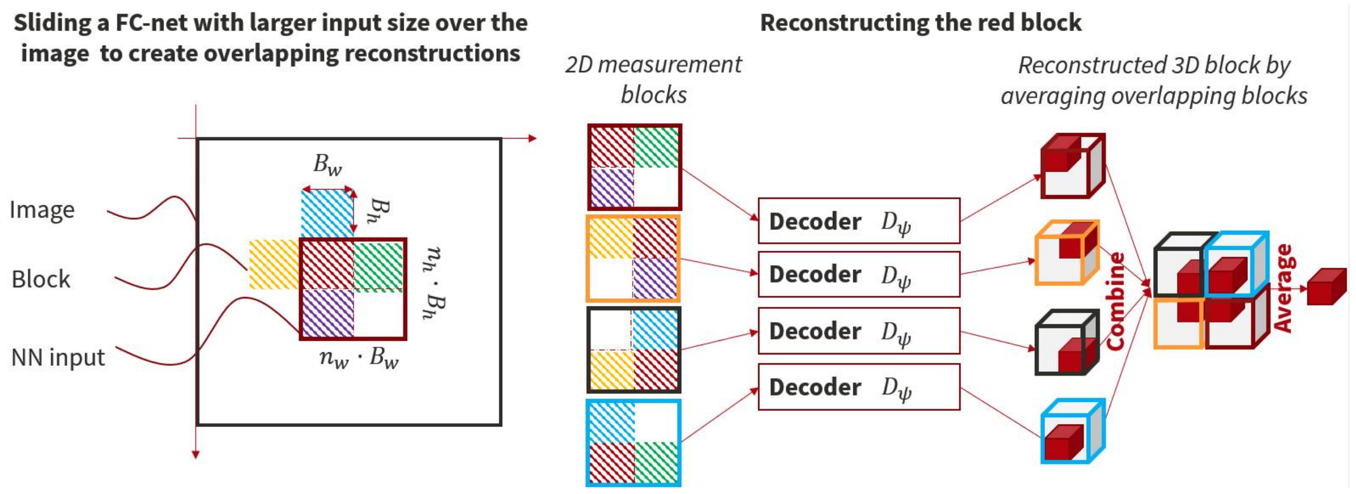
For all the training experiments with 3D U-Nets, a video-block used as a training sample is produced as follows: A $256 \times 256 \times N$ (with N the number of slots of the shutter function) video-block is extracted from the video-clip. The original size of video clips in the NfS dataset is $1280 \times 720 \times F$ (with F, the number of frames in a given video clip, those are ranging from a few hundreds to a few thousands). The first frame of the video-block is randomly sampled within the length of the video-clip $i \in \{0, ...F-N\}$. Then a scale $s \in [0.36, 0.40]$ is randomly sampled to rescale the video-clip. Note that the minimum scaling factor of 0.36 is chosen so that 0.36×720 is slightly larger than 256, while the maximum scaling is chosen arbitrarily so that the maximum zoom level roughly matches the typical feature size one expects to generalize to for reconstruction. The N frames in the video block are then cropped to a size of 256×256. The $256 \times 256 \times N$ block we obtain is then randomly flipped horizontally and vertically, and is randomly time-reversed, in addition a random gamma adjustment of brightness is performed with $\gamma \sim \mathcal{N}(1., 0.3)$.

For all the training experiments with FC-nets, the same procedure applies but video-blocks of size $n_w \cdot B_w \times n_h \cdot B_h \times N$ are sampled, in which $(n_w \times n_h)$ is the number of blocks averaged in the reconstruction of a single block and $B_w \times B_h$ is the number of free parameters of optimized shutter function (using FC-net we only learn pixel-wise independent shutter functions within a block). More specifically, our FC-nets are trained on blocks of size $n_w \cdot B_w \times n_h \cdot B_h \times N$ which allows the reconstruction at inference to slide the FC-net of input size $n_w \cdot B_w \times n_h \cdot B_h \times N$ by a stride of B_w and B_h in each spatial dimension. Thus, we obtain, for each block (leaving out the borders for the sake of clarity) $(n_w \times n_h)$ reconstructions that can be averaged thus avoiding blocky artifacts in the reconstruction. This procedure is similar to the one proposed in [1] and is further illustrated in Figure 1.

1.2 HDR 2D U-Net

The 2D U-Net we use takes as an input $H \times W = 256 \times 256$, a low dynamic range coded image and produces a 256×256 high-dynamic range image. The architecture we use for our HDR reconstructions consists of 6 "downsampling modules" and 6 "upsampling modules". Each downsampling module by m consists of a convolution $C^{\downarrow}_{m,1}$ with N_m 3×3

1. http://hdrihaven.com
2. http://ci2cv.net/nfs/index.html



Sliding a FC-net with larger input size over the image to create overlapping reconstructions
Image
Block
NN input
B_w
B_h
$n_h \cdot B_h$
$n_w \cdot B_w$
Reconstructing the red block
2D measurement blocks
Decoder D_ψ
Decoder D_ψ
Decoder D_ψ
Decoder D_ψ
Reconstructed 3D block by averaging overlapping blocks
Combine
Average

Fig. 1. **Illustration of the reconstruction by blocks using FC-nets.**

kernels followed by a ReLU, followed by another convolution $C^{\downarrow}_{m,2}$ with N_m 3×3 kernels, followed by another ReLU. Both the convolutions have a striding of 1 and a padding of 1. The output of a module in the downward pass is then downsampled by 2 using average pooling. We have $N_m = 2^m \cdot 32$, thus at the most downsampled U-Net level we have $N_5 = 1024$ feature maps. Each "upsampling module" consists of a bilinear upsampling by 2 followed by a convolution $C^{\uparrow}_{m,1}$ with 3×3 kernels, this is concatenated to the output of the corresponding "downsampling module", the concatenated input is then followed by a double convolution module analog to the one in the downward pass.

Training loss and hyper-parameters: We train the HDR 2D U-Net architecture with ADAM (with ($\beta_1 = 0.99$ and $\beta_2 = 0.999$, without weight decay) a learning rate of $\eta_\psi = 1 \cdot 10^{-3}$ and a batch size of 32, the encoder is learnt with a slower learning-rate of $\eta\theta = 0.1 \cdot \eta_{psi}$. A gradient clipping of 0.1 is used. We tested both a L1-loss on the HDR values as well as a L1-loss on the tone-mapped values, PSNR and SSIM do not reflect significant differences between the two in our experiments, even though the type of errors they produce is different. Notably the L1-loss on the direct HDR values enforces bright regions (with high values) to be more correct.

1.3 Video Compressive Sensing FC-nets

The FC-net we use takes as an input multiple $(n_h \times n_w)$ blocks of size $B_h \times B_w$ and produces $n_h \times n_w$ video-blocks of size $B_h \times B_w \times N$. To reconstruct a whole image the input of the FC-net is slid by a stride of (B_h, B_w) in both spatial dimensions so that all blocks (ignoring blocks on the borders for clarity) get reconstructed $n_h \times n_w$ times and averaged (each reconstruction contains information coming from different neihbourhoods) as illustrated in Figure 1. As mentioned earlier, this scheme is used to avoid blocky artifacts. The FC-net in our experiments has $n_w = n_h = 2$, $B_h = B_w = 8$, thus the input to the neural network is of size $n_w \cdot n_h \cdot B_h \cdot B_w = 256$, we use $N = 16$ slots (which is also the number of reconstructed frames thus the output is of size $n_w \cdot n_h \cdot B_h \cdot B_w \cdot N = 256 \cdot 16 = 4096$. Each block thus benefits from $n_h \cdot n_w = 4$ reconstructions and the FC-net has 3 hidden layers with 4096 units. All in all, this is the following FC-net architecture $256 \times 4096 \times 4096 \times 4096$.

Training loss and hyper-parameters: We train the HFR FC-net architecture with ADAM (with ($\beta_1 = 0.99$ and $\beta_2 = 0.999$, without weight decay) a learning rate of $\eta_\psi = 1 \cdot 10^{-4}$ and a batch size of 256, the encoder is learnt with the same learning-rate of $\eta\theta = \eta_{psi}$. A gradient clipping of 1 is used. We use a voxel-wise (space-time) L1-loss.

1.4 Video Compressive Sensing 3D U-Net

The 3D U-Net we use takes as an input a $H \times W = 256 \times 256$ coded image and produces a $H \times W \times N = 256 \times 256 \times 16$ video-clip. The architecture we use for our high-speed reconstructions now consists of 5 "downsampling modules" and 5 "upsampling modules". Similar to the 2D U-Net architecture we use for HDR imaging. Each downsampling module by m consists of a convolution $C^{\downarrow}_{m,1}$ with N_m $3 \times 3 \times 3$ 3D-kernels followed by a ReLU, followed by another convolution $C^{\downarrow}_{m,2}$ with N_m $3 \times 3 \times 3$ kernels, followed by another ReLU. Both the convolutions have a striding of 1 and a padding of 1. The output of a module in the downward pass is then downsampled by 2 using a 3D-average pooling. We have $N_m = 2^n \cdot 48$, thus at the most downsampled U-Net level we have $N_4 = 768$ feature maps. Each "upsampling module" consists of a bilinear upsampling by 2 followed by a convolution $C^{\uparrow}_{m,1}$ with 3×3 kernels, this is concatenated to the output of the corresponding "downsampling module", the concatenated input is then followed by a double convolution module analog to the ones in the downward pass.

Training loss and hyper-parameters: We train the HFR 3D U-Net architecture with ADAM (with ($\beta_1 = 0.99$ and $\beta_2 = 0.999$, without weight decay) a learning rate of $\eta_\psi = 5 \cdot 10^{-4}$ and a batch size of 256, the encoder is learnt with a slower learning-rate of $\eta\theta = 0.1 \cdot \eta_{psi}$. A gradient clipping of 1 is used. We use a voxel-wise, that is in space-time, L1-loss.

1.4.0.1 **Training stopping criterion**: All our FC-Net models are trained for a maximum of 150 epochs. All our U-Net models are trained for a maximum of 100 epochs. Those numbers were observed to be the start of overfitting. Those

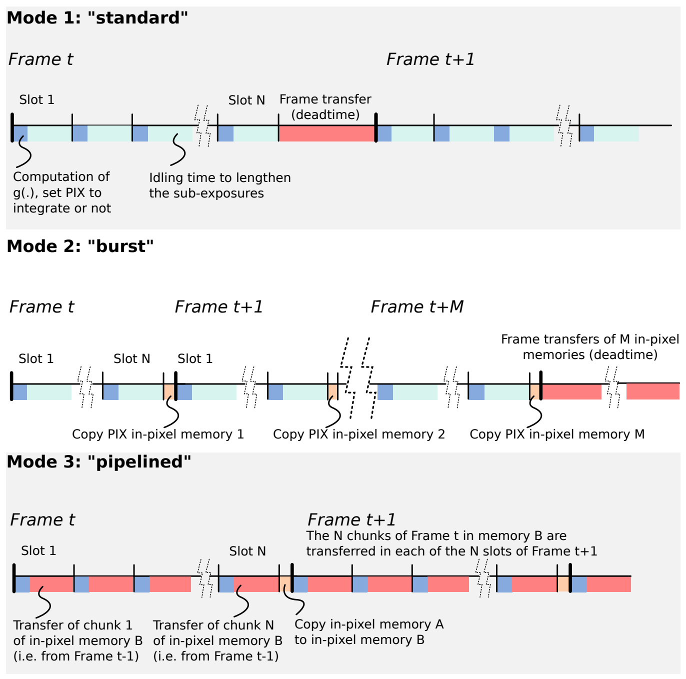

Mode 1: "standard"
Frame t
Frame t+1
Slot 1
Slot N
Frame transfer (deadtime)
Computation of g(.), set PIX to integrate or not
Idling time to lengthen the sub-exposures
Mode 2: "burst"
Frame t
Frame t+1
Frame t+M
Slot 1
Slot N
Slot 1
Frame transfers of M in-pixel memories (deadtime)
Copy PIX in-pixel memory 1
Copy PIX in-pixel memory 2
Copy PIX in-pixel memory M
Mode 3: "pipelined"
Frame t
Frame t+1
The N chunks of Frame t in memory B are transferred in each of the N slots of Frame t+1
Slot 1
Slot N
Transfer of chunk 1 of in-pixel memory B (i.e. from Frame t-1)
Transfer of chunk N of in-pixel memory B (i.e. from Frame t-1)
Copy in-pixel memory A to in-pixel memory B

Fig. 2. **Illustration of the different transfer models we implemented on our sensor-processor**

we determined using a subset of the training data used as a held-out validation set and monitoring whenever the error on this validation set was starting to increase.

2 Additional details about the optimized shutter functions and their implementations

2.1 Parametrization of the shutter functions

All the shutter functions we train in our work (classes (b) to (e)) are defined over an integer domain: the parameters (the slot numbers) are integers. In addition, their co-domain is binary: the shutter is either "on" or "off". Those are constraints that need to be enforced during the training of the encoder. In simulation, we decompose the functions $f(\cdot)$ in elementary functions that are shifted and horizontally flipped variants of the Heaviside functions, hence the output is binary. The domain is ensured to be integers by rounding operations. Then, using the automatic differentiation in Pytorch [2] we solely have to implement the backward pass of the Heaviside function (used as an elementary block) and of say, the flooring operator (rounding and ceiling are also simple variants of flooring).

2.2 Implementations of the shutter functions on our sensor-processor

The implementation of the shutter functions on our SCAMP-5 sensor-processor follows closely Algorithm 1 in the main text. Nevertheless, we omitted details about the readout of frames from the device. SCAMP-5 can carry-out computations while integrating light, and it is also possible to perform a frame readout when SCAMP-5 integrates light. One can think of the value of the light-register, that we denote PIX, as changing as light falls onto it. While we implement the shutter functions, the value of PIX is reset to 0 at the end of each slot after being either added to the frame register when the shutter is "on" or discarded when the shutter function (the output value of $g(\cdot) = 0$) is "off".

Integration of light thus can always happen in the background whether a frame is being readout, or whether computation happens. However, computation and frame readout are mutually exclusive. This implies that a pixel shutter cannot be modulated while a frame is being readout. This means, in a **"standard"** readout mode, as illustrated in the first row of Figure 2, there is a dead-time at the end of each frame, when it is being readout. To mitigate this issue we implemented two other transfer modes.

In case one wants to image a transient and very fast phenomenon without a deadtime every coded frame, one can delay the transfer using in-pixel memory buffers. If one uses M in-pixel registers to do so, one needs to transfer M frames every M frames. We were able to keep $M = 4$ out of the 7 analog registers available on SCAMP, and we can thus take a **"burst"** of 4 coded images without deadtime in this mode. This is illustrated in the second row of Figure 2.

Finally, we implemented a third mode in which a double-buffer mechanism is used, along with pipelining, to mask the transfer/readout time. To do so, each frame is divided into N sub-frames, which are sent out in chunks. Specifically, since it is possible to integrate light while transferring a frame, a chunk containing a sub-frame is sent out in each slot, in place of idling the processor to lengthen the sub-exposures. Hence, the number of sub-frames and chunks have to match. Note that, the value of a pixel in the coded frame is only fully determined at the end of the final Nth slot. Thus, the pipelining we have just described can only be performed on a coded frame that has been fully captured. At the end of the Nth slot of a frame t, instead of being readout, this frame is copied in an in-pixel memory A, this is the frame that will be sent in chunks during the formation of the coded frame $t + 1$ in another memory, say B. We illustrate this **"pipelined"** transfer mode in the third row of Figure 2.

2.3 Remarks about the optimized shutter function

class (b): The learning of class (b) converges to a mostly gray 50% exposure level and uses low exposures distributed in a "pseudo" poisson-disk fashion allowing it to reconstruct very high irradiances. This is because we introduced a gain parameter (a single variable for every HDR) optimized by the network for each image (as part of the encoder), so that we can accomodate many different illumination conditions in simulation. In the real-world experiments, this "gain" is ignored. However, its analog is that each image is taken so as to have most of the image well-exposed (as one reasonable photographer would do). The learnt gain in simulation thus translates to a global exposure time in the real-world experiment. This global exposure was configured using the exact same mechanism as the one discussed in Section 3.2).

class (c): Class (c) patterns have a single degree of freedom allowing them to shift their "offset", that is when the pixel start exposing, in time. Through learning, pixels tend to make sure they cover different starting times. Hence one can see in Figure 7 that clusters that were initially present in the randomization are spread-out.

class (d): Due to the varying exposure lengths across pixels, class (d) seems to pose an issue for reconstruction. The deep networks we trained to reconstruct those codes converged much later and some of them still presented artifacts such as flickering in some regions of the videos or with regions showing a clearly wrong average brightness. Some authors in the compressive sensing literature have noted the importance of 1) sampling all the pixels an equal number of times in the random projections and 2) each pixel capturing with random projections of "equal amount" of energy. Clearly, assumption 2) is violated, and as it can be noted in the images, the codes through learning indeed tend to converge towards having equal lengths.

class (e): The learning of class (e) tends to converge towards lengthened exposure bumps with much fewer low frequencies (in the time dimension) than a random pattern.

3 Other experimental considerations

This section gives experimental details about the capture of real data using our SCAMP-5 prototype.

3.1 About our prototype sensor-processor

SCAMP-5 is used as our prototype programmable sensor-processor for all our experiments. SCAMP-5 is a 256×256 pixel array with mixed-signal ALUs collocated with photo-sensitive elements in each pixel. Each processing element contains 7 analog memories as well as a 13 1-bit single bit registers. Those are implemented using 176 transistors, yielding a pixel pitch of about $30\mu\text{m}$ with a fill-factor of about 6% in a $0.18\mu\text{m}$ 1P6M CMOS process. Additional details about the sensor can be found in [3].

3.2 Setting the exposure duration

The sub-exposure lengths (i.e the duration of each time slot) is configurable in our prototype system.

High-speed imaging: In all our experiments the sub-exposure lengths were varied to match the physical time scale of the process being imaged. Those range from less than 0.3ms ($>$ 3000FPS) (balloon experiment) to about a millisecond. They are controlled by a single parameter that arbitrary "lengthens" δt (refer to Algorithm 1 in the main text) by idling the pixel-processors at the end of their processing. Note that the pixel-wise computation time is negligible for all codes because the decoding functions $g(.)$ functions are designed to be trivial to compute on-chip.

High-dynamic range imaging: In the high-dynamic range experiment we made sure that all the pixels are globally-well exposed, as we mentioned earlier in Section 2.3.

The total exposure-length for a given frame is solely determined by the number of sub- exposures, which is set to 16 in all the high-speed experiments and 64 to all the HDR experiments, multiplied by their length, that is 0.3 to $\sim 1\text{ms}$ and about 5ms in the HDR experiments.

	PSNR	SSIM
Bilinear	18.95	0.4133
ThinnedOut	20.65	0.4556
Hitomi et al.	26.22	0.7258
FC-net (ours, opt. class (c))	**38.14**	**0.9555**
U-Net (ours, opt. class(c))	32.40	0.8118

TABLE 1
Summary of additional baselines with 200 video-clips taken from the NFS dataset.

3.3 Artificial lighting for high-speed imaging

Due to the very short integration times that were used in some of our high-speed imaging setups, and due to the low fill factor of our sensor-processor prototype (about 6%), some of our experiments (for instance, the exploding balloon) were performed under artificial lighting (using an ARRI Junior 1000 Plus Tungsten Fresnel light controlled by a potentiometer). The irradiance we measured in our artificial lighting scenarios is typically between $1.1 - 1.7\text{kW}\,\text{m}^{-2}$ measured at 561nm (lower bound is comparable to the solar power density measured for a clear sky on a bright day at sea-level).

4 Additional Results

4.1 Video Compressive Sensing / High-speed imaging

We provide additional results in the form of qualitative and quantitative results illustrating the discrepancies between our method and baselines in Figure 3 and Table 1.

The bilinear baseline refers to a reconstruction obtained from the coded exposure of a frame by bilinearly interpolating all the pixels that had their shutter off at the time of the reconstructed frame.

The thinned out baseline refers to a reconstruction obtained from a ground-truth frame by subsampling in space by the same amount the final reconstruction is super-resolved in time. In the case of a 16 slot shutter function inducing a $16\times$ super resolution, the ground truth frame to be thinned out is subsampled by 4 in each spatial direction. We use bilinear downsampling and bilinear interpolation for the resizing of the full frame.

The Hitomi et al. baseline is an equivalent of the method for video compressive sensing and reconstruction presented in [4], it is using a dictionary with 4096 atoms learnt with orthogonal matching pursuit (OMP). The reconstruction also uses orthogonal matching pursuit. The dictionnary is learnt on 500000 patches extracted from the same training set we used for learning our U-Nets and FC-nets (refer to Section 1).

For Table 1, we used 200 video-clips randomly extracted from the same validation set we used to evaluate our U-Nets and FC-nets

References

[1] M. Iliadis, L. Spinoulas, and A. K. Katsaggelos, "Deep fully-connected networks for video compressive sensing," *Digital Signal Processing*, vol. 72, pp. 9–18, 2018.

[2] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

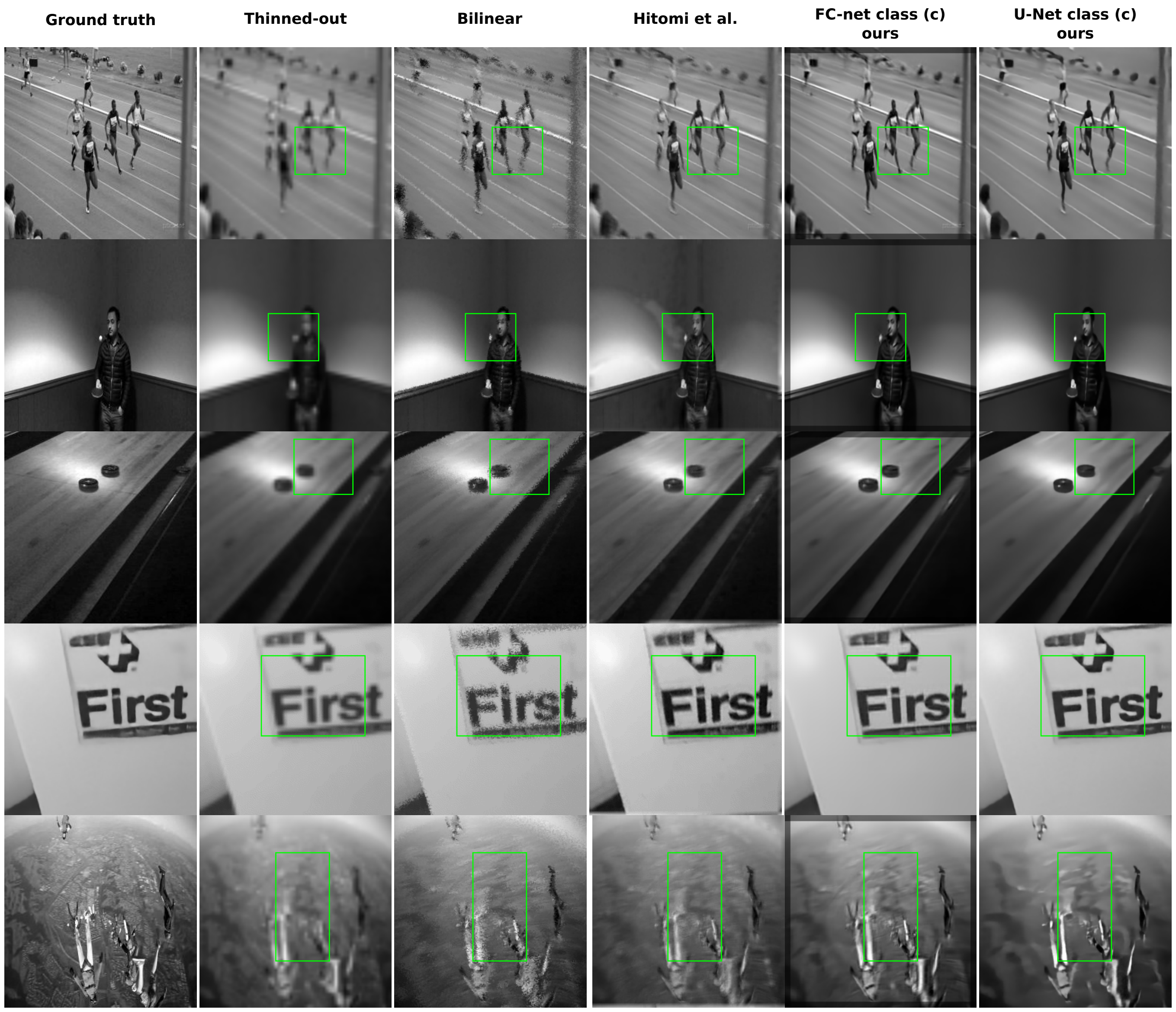


Fig. 3. **A visual quality comparison of our results against other baselines.** The pannel is best viewed in an electronic document viewer while zooming-in. Green squares emphasize some of the regions where the differences are most noticeable between the methods.

[3] S. J. Carey, A. Lopich, D. R. Barr, B. Wang, and P. Dudek, "A 100,000 fps vision sensor with embedded 535gops/w 256× 256 simd processor array," in *2013 Symposium on VLSI Circuits*. IEEE, 2013, pp. C182–C183.

[4] Y. Hitomi, J. Gu, M. Gupta, T. Mitsunaga, and S. K. Nayar, "Video from a single coded exposure photograph using a learned over-complete dictionary," in *Int. Conference on Computer Vision (ICCV)*. IEEE, 2011, pp. 287–294.