Efficient Real-Time Raw-to-Raw Denoising for Extreme Low-Light Ultra HD Video on Mobile Devices

Supplementary Material

Anonymous CVPR submission

Paper ID

S.1. Overview

This supplementary document provides additional details that did not fit into the main paper due to space constraints. In particular, we cover:

1. Mobile captured extreme low-light videos with and with out the proposed solution integrated into the mobile camera pipeline (Section S.2).
2. Additional qualitative visual comparisons (Section S.3).
3. Additional implementation details for the synthetic and real data pipelines (Section S.4).
4. Detailed structural re-parameterization (layer fusion) (Section S.5).
5. Detailed spatial resolution reduction (restructuring) (Section S.6).
6. Mobile on-device integration/porting procedure (Section S.7).

Unless otherwise noted, the notation and experimental setup follow the main paper.

S.2. Mobile captured extreme low-light videos

we have captured extreme low light scenes in an Android mobile with and with out the model integrated to ISP [1]. Due to space constraint we include cropped videos (noisy and different model outputs) in the supplementary zip file for evaluation of the solution developed using proposed methodology.

S.3. Additional qualitative visual comparisons

Additional results of impact of synthetic data have been depicted in Figure 1, 2, and 3. Additional results of visual comparisons between differe low-light video denoising models have been shown in Figure 4 and 5.

[1] Gain is applied in ISP to make the extreme low-light scenes visible.

S.4. Synthetic and real data pipelines

S.4.1. Synthetic bright blob addition

Presence of bright light source in dark scenes creates color artifacts around edges (example shown in Figure 6). To mimic such light source in dark scene, we added bright, random shaped blobs of different colors (red, green, blue, white) to scaled dark images. The degradation pipeline with blob addition block has been shown in figure 8. Example synthetic dark images (Set1) with added blobs have been shown in Figure 7.

S.4.2. Synthetic noise vs real capture noise

We use heteroscedastic Gaussian distribution to model the noise under extreme low light conditions. The model hyperparameters were empirically chosen to match the noise close to the actual extreme low-light capture noise. One example of synthetic generated noisy data and real noisy captured under low-light has been shown in Figure 9.

S.4.3. Real world dataset capture process

Real db preparation: Our dataset was prepared in a dedicated lab enabling precise control of environmental intensity and scene variations. Scene lux was measured using a lux meter at four corners and averaged. Data capture under $<$1lx conditions utilized an Android device mounted on a tripod to prevent perturbations, with static scenes ensuring no local motion. Two high-frequency light sources are used to eliminate light flickering.

Videos were recorded at 30fps, and 90 consecutive RAW frames (GBRG Bayer format) per video were selected after discarding initial/terminal frames. These frames were averaged to create a denoised base, followed by residual noise removal using a 16-layer mRLFB model.

From 160 captured videos, all GTs underwent manual quality checks. Sixty were discarded due to quality issues (e.g., shadow motion, lighting changes), retaining 100 high-

Figure 1. **Impact of Synthetic Data.** Visual comparison of base model outputs trained on Synthetic (Set1,Set2, combined), Real-World Static, and Hybrid (pre-trained on Synthetic + fine-tuned on Real-World Static)

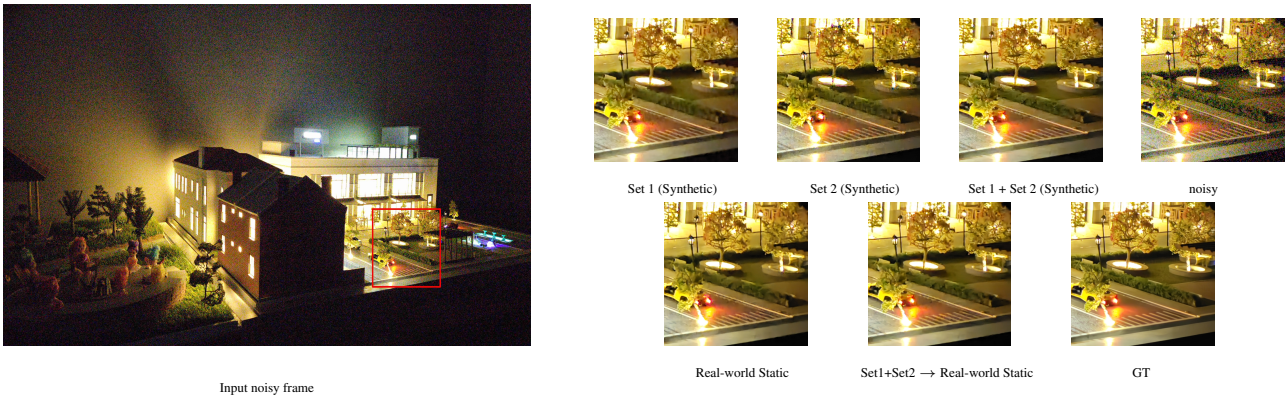

Figure 2. **Impact of Synthetic Data.** Visual comparison of base model outputs trained on Synthetic (Set1,Set2, combined), Real-World Static, and Hybrid (pre-trained on Synthetic + fine-tuned on Real-World Static)

quality sequences

Motion DB Preaparation: We created a motion vector dictionary by capturing videos (handheld, panning, zooming in and out, rotation, etc.) under normal lighting. Static scenes were filmed with a moving camera to avoid local motion. Motion vectors (one per consecutive frame pair) were estimated using rigid transformations (avoiding complex homography to preserve noise characteristics) and stored. Vectors are randomly selected from the dictionary and applied to neighboring frames in static videos.

S.5. Structural re-parameterization (layer fusion)

We hereby describe the algorithm used to compress the multi-skip conv block to a single 3x3 conv layer. Consider the following notation: - $C_{in}^{(k)}$ and $C_{out}^{(k)}$ denote the number of input and output channels to the conv layer of shape $k \times k$ - C_{in} denotes the number of input channels to the multi-skip block - C_{out} denotes the number of output channels to the multi-skip block - M_1 and M_2 denote the number of intermediate channels in the multi-skip block. - The 3x3 kernel has 9 spatial positions, indexed as $s = 1, 2, 3...9$ - Weights of the 3x3 convolution layer are denoted by $W^{(3)}$ having shape $C_{in}^{(3)} \times 3 \times 3 \times C_{out}^{(3)}$. - $W_s^{(3)}$ represents each slice of $W^{(3)}$ across the spatial domain, having shape $C_{in}^{(3)} \times 1 \times 1 \times C_{out}^{(3)}$. - Similarly, the 1x1 convolution weights are denoted by $W^{(1)}$ having size $C_{in}^{(1)} \times 1 \times 1 \times C_{out}^{(1)}$ - The bias vectors are denoted by $b^{(1)}$ of length $C_{out}^{(1)}$ and $b^{(3)}$ of length $C_{out}^{(3)}$

Case 1: $1 \times 1 \rightarrow 3 \times 3$ In this case, $C_{in}^{(1)} = C_{in}, C_{out}^{(1)} = M_1, C_{in}^{(3)} = M_1, C_{out}^{(3)} = M_2$ Now, the fused kernel will be single 3x3 kernel effectively having $C_{in}^{(3)} = C_{in}, C_{out}^{(3)} = M_2$. Each slice of the fused kernel weights can be defined as

$$W_s^{fused} = W^{(1)} W_s^{(3)}, s = 1, 2...9$$

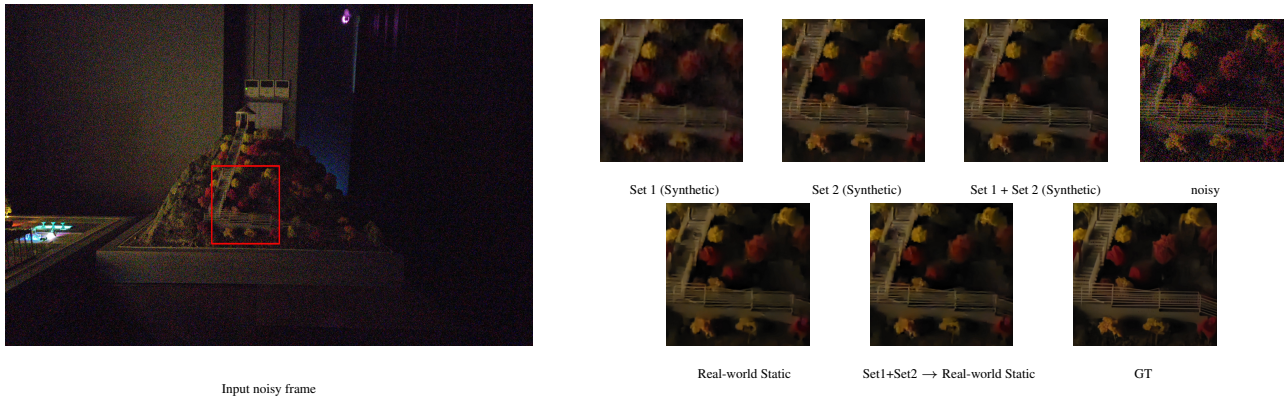

Figure 3. **Impact of Synthetic Data.** Visual comparison of base model outputs trained on Synthetic (Set1,Set2, combined), Real-World Static, and Hybrid (pre-trained on Synthetic + fine-tuned on Real-World Static)

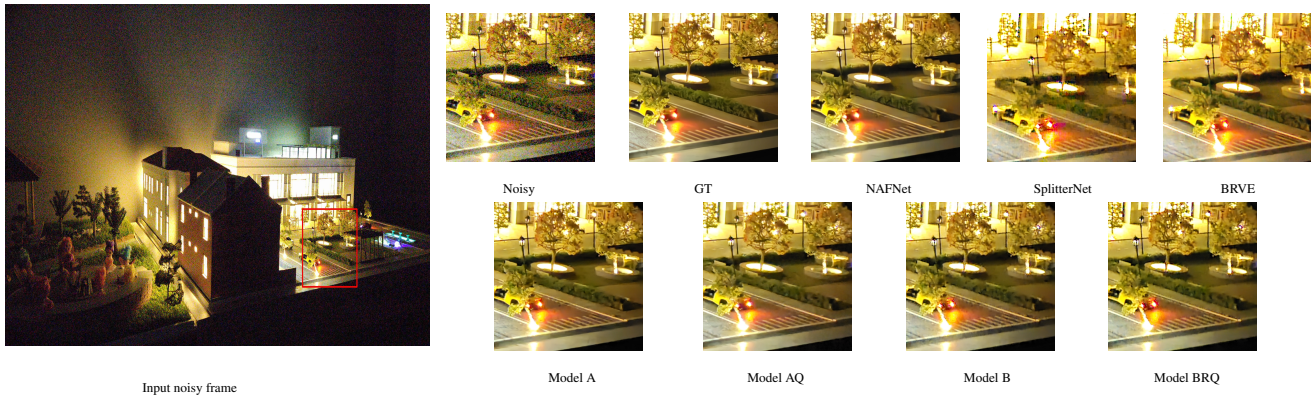

Figure 4. Visual comparison of different low-light video denoising methods on Real world captures

$$b^{fused} = b^{(3)} + \sum_{s=1}^{9} W_s^{(3)} b^{(1)}$$

On the other hand, when we combine a 3x3 kernel followed by a 1x1 kernel, the kernel weights calculation is the same except the order is reversed, whereas the bias calculation is defined as

$$b^{fused} = b^{(1)} + W^{(1)} b^{(3)}$$

Case 2: $k \times k$ + skip A skip connection can be thought of as a parallel branch of the re-parameterizable block having a $k \times k$ kernel with identity weights i.e. the input and output to the kernel are exactly the same. The weights of such a kernel can be defined as

$$W_{identity}[u, v, i, j] = \begin{cases} 1, & \text{if } u = \left\lfloor \frac{k}{2} \right\rfloor,\ v = \left\lfloor \frac{k}{2} \right\rfloor,\ i = j, \\ 0, & \text{otherwise.} \end{cases}$$

The bias vector will be a zero vector of length C_{out}. Considering the two kernels in parallel, the combined weights can be obtained by simply adding the two.

$$W_{combined} = W_{k \times k} + W_{identity}$$

$$b_{combined} = b_{k \times k}$$

S.6. Spatial Resolution Reduction (restructuring)

Modern NPUs on mobile devices exhibit peak efficiency when executing convolution layers whose input and output channel dimensions are aligned with their internal vectorization patterns. Let a convolution layer be parameterized by a kernel tensor

$$\mathbf{W} \in \mathbb{R}^{k \times k \times C_{\text{in}} \times C_{\text{out}}}$$

and let the input feature map be

$$\mathbf{X} \in \mathbb{R}^{H \times W \times C_{\text{in}}}.$$

Empirically, the most optimized configurations correspond to $C_{\text{in}} = 32$ and $C_{\text{out}} = 32$. These settings align well with

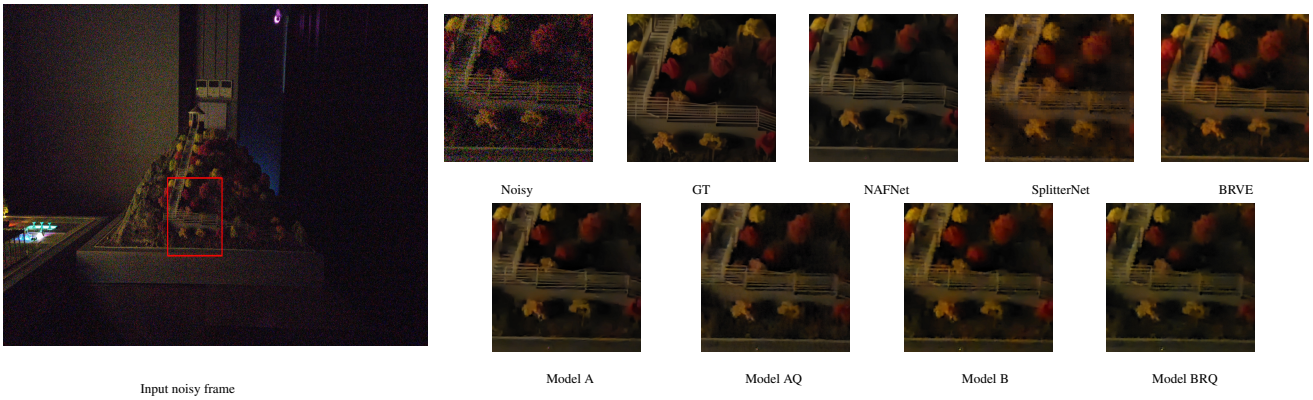

Figure 5. Visual comparison of different low-light video denoising methods on Real world captures

Figure 6. **Color artifact.** Prevalent around edges of bright light source in dark surroundings. These artifacts occur due to incorrect modification of pixels around edges of light source in the denoising process.

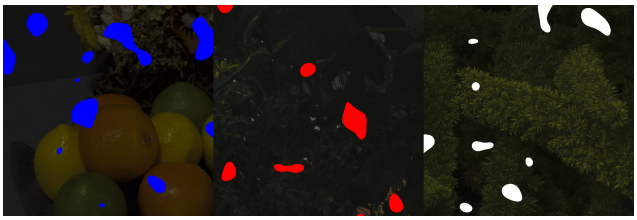

Figure 7. **Synthetic images with color blobs.** Color blobs added to the scaled dark images to replicate light source with dark surroundings. This data in training set helped removing the color artifacts around bright source edges.

the NPU's internal SIMD width, allowing the hardware to process channel blocks efficiently. Interestingly, for layers operating at the same spatial resolution, we observe that the runtime for $16 \times k \times k \times 16$ convolutions matches that of $32 \times k \times k \times 32$ convolutions.

Motivated by this behavior, we consider a model restructuring strategy in which the network is trained using a compact topology characterized by intermediate feature maps of resolution $\frac{H}{4} \times \frac{W}{4} \times 16$. Representing feature maps as

$$\mathbf{F} \in \mathbb{R}^{\frac{H}{4} \times \frac{W}{4} \times 16},$$

this architecture maintains efficient memory usage and computational cost while preserving the representational capacity needed for high quality restoration.

The core idea enabling restructuring is the local and translation invariant nature of convolution. A convolution layer with kernel

$$\mathbf{W} \in \mathbb{R}^{k \times k \times C_{\text{in}} \times C_{\text{out}}}$$

can be transformed into another mathematically equivalent layer

$$\mathbf{W}^* \in \mathbb{R}^{k \times k \times 2C_{\text{in}} \times 2C_{\text{out}}}$$

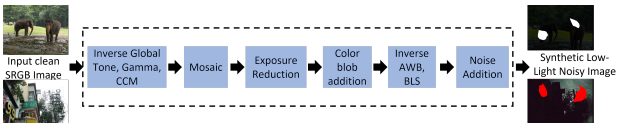

Figure 8. **Synthetic data generation pipeline.** Color blobs were added after exposure reduction to mimic bright light source in dark background.

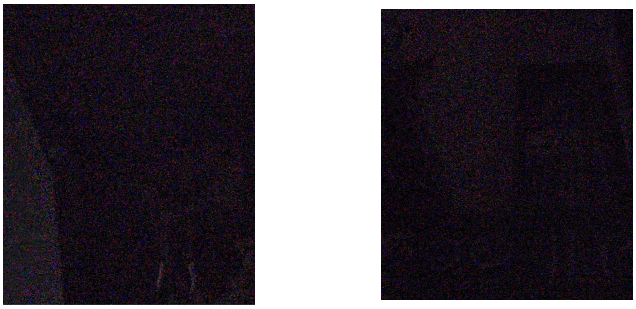

Figure 9. **Noise comparison.** on the left original noisy patch and on the right synthetic noisy patch (**move to supplementary**)

provided the associated input and output feature maps are rearranged accordingly. Because convolution acts independently across spatial locations, such expansions correspond to deterministic permutations and groupings of channels that preserve functional equivalence. However, these transformations require exact coordination between how feature maps are reordered and how kernels are tiled, making correct reshaping essential to preserve the behavior of the original model.

We modify the front end of the architecture to incorporate this change. The original design employs a depth to space operator that rearranges the raw input image into channel wise feature maps. Let the raw input be

$$\mathbf{I} \in \mathbb{R}^{H \times W \times 1}.$$

Depth-to-space with block size 4×4 yields

$$\mathbf{F} * \text{d2s} \in \mathbb{R}^{\frac{H}{4} \times \frac{W}{4} \times 16}.$$

We replace the depth-to-space operator with a non-learnable convolution layer using stride 4 in height and 8 in width with the following weights.

$$w[i, j, 0, k] = \begin{cases} 1, & k = 8i + j, \\ 0, & \text{otherwise.} \end{cases}$$

The resulting feature map

$$\mathbf{F} * \text{conv} \in \mathbb{R}^{\frac{H}{4} \times \frac{W}{8} \times 32}$$

is identical to that produced by depth to space, differing only by a fixed interleaving of channels. This substitution retains the structural intention of the original design while enabling subsequent restructuring steps to operate in a unified convolutional framework.

A natural question is why one should reshape a trained network rather than train the restructured architecture from scratch. We observe that directly training the expanded topology leads to degraded image quality due to an induced directional bias. Because the restructured model effectively increases the receptive field more along the horizontal direction, training from scratch tends to emphasize horizontal dependencies, reducing the ability to reconstruct vertical edges and fine textures. This results in noticeable loss of detail in vertically oriented structures. In contrast, applying a mathematically equivalent transformation to a fully trained, unbiased model preserves its learned distribution of features while enabling structural compatibility with the new topology. This ensures that the representational anisotropy introduced by restructuring does not distort the learned filters.

Once the initial feature map layout is defined, we propagate the channel interleaving pattern through the entire network. Each convolution layer is reshaped by reorganizing its kernels according to the order of channels established at the input stage. The resulting model maintains the functional behavior of the original network while adopting a topology that is power efficient, runtime optimized, and aligned with NPU friendly dimensions. This structured approach allows the deployment of high quality models on mobile devices without sacrificing accuracy or increasing computational load.

S.7. Mobile on-device integration/porting procedure (Section)

We deploy our trained models on-device using Qualcomm's Snapdragon Neural Processing Engine (SNPE). Each model is converted into a mobile-optimized Deep Learning Container (DLC), enabling execution directly on the Neural Processing Unit (NPU) using shared buffers (ION).

The model receives the raw sensor readout in GBRG Bayer format. The generated output is consumed by the downstream ISP pipeline for demosaicing, tone mapping, gamma correction, and other image formation steps.