A. Implementation details for U-BSN architecture

The U-BSN architecture begins with the input undergoing two 3×3 convolutions before entering the encoder. The encoder then repeatedly applies 3×3 convolutions at each layer, followed by a ReLU activation and a max-pooling operation with a stride of 2 for downsampling. Unlike traditional approaches where the number of channels doubles at each downsampling step, we maintain a consistent 128 channels across all layers to focus more effectively on nearby features. Additionally, to increase the model's capacity to process high-resolution information, extra 3×3 convolution layers are added – five in the first level and three in the the second level.

The decoder substitutes 3×3 convolutions with 1×1 convolutions to efficiently aggregate multi-scale information with lower computational overhead while maintaining $\mathcal{J}$-invariance. It repeats the process of upsampling, followed by channel-wise concatenation with the corresponding feature map from the skip connection, which has passed through a blind spot convolution layer. This feature then undergoes a 1×1 convolution and a ReLU activation. The final output is refined through a series of four 1×1 convolutions.

B. Acquisition of effective receptive field

The effective receptive field (ERF) quantifies the influence of each input pixel location on the central output value, defined as $\frac{\partial y_{0,0}}{\partial x_{i,j}}$, where $y = f_\theta(x)$ and $f_\theta(x)$ is a pretrained network. To calculate the ERF, we first set the error gradient $\frac{\partial l}{\partial y_{0,0}} = 1$ and $\frac{\partial l}{\partial y_{0,0}} = 0$ for all $i \neq 0$ and $i \neq j$. We then perform backpropagation to compute the gradients.

For this study, we used networks pretrained on the SIDD dataset. Since the ERF is input-dependent, we averaged the ERF over images from the SIDD validation set to obtain a representative measure. ERFs are visualized using the 1st and 99th percentiles.

C. Measure efficiency

Training time & Memory Training time refers to the duration required for one gradient update. We measured the training time over $1,000$ gradient steps and divided the total time by $1,000$. The input data dimension was set to $(1, 3, 256, 256)$. Memory usage was monitored during training.

MACs & Number of parameters The number of multiply-accumulation operations (MACs) and the total number of parameters were calculated using `thop` [38] package with an input size of $(1, 3, 256, 256)$.