Appendix

A. Implementation Details

In this section, we introduce in detail the implementations of ACDA. We use PyTorch for all the implementations. In practice, the per-pixel adaptive convolutions can be efficiently implemented by first reorganizing input features with the *Unfold* function in PyTorch to extract sliding local blocks from a batched input tensor. And then perform corresponding multiplications between sliding local blocks and the adaptive filters (dynamic atoms). All experiments are conducted on the same machine with 4×2080Ti GPUs. The code will be released upon acceptance.

A.1. Visualization of filter decomposition

We provide a visualization of decomposing a convolutional filter K over $m = 4$ basis elements in Figure A.

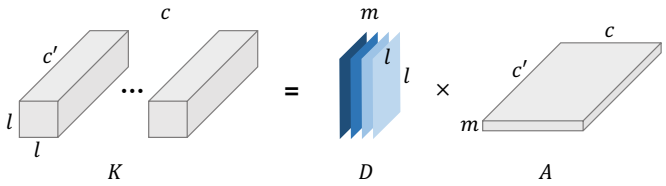


Figure A: Illustration of decomposed convolutional filters with $m = 4$ dictionary atoms. A convolutional filter $\mathbf{K} \in \mathbb{R}^{c\times c'\times l\times l}$ is decomposed over m filter atoms $\mathbf{D} \in \mathbb{R}^{m\times l\times l}$, linearly combined by coefficients $\mathbf{A} \in \mathbb{R}^{c\times c'\times m}$.

A.2. Visualization of Multi-scale Fourier-Bessel Bases

Fourier-Bessel bases at three scales are visualized in Figure B. In practice, we truncate the first 6 basis elements at each scale. The small-size bases are padded with 0 to match the spatial size of the largest bases, which allow parallel multiplications and convolutions.

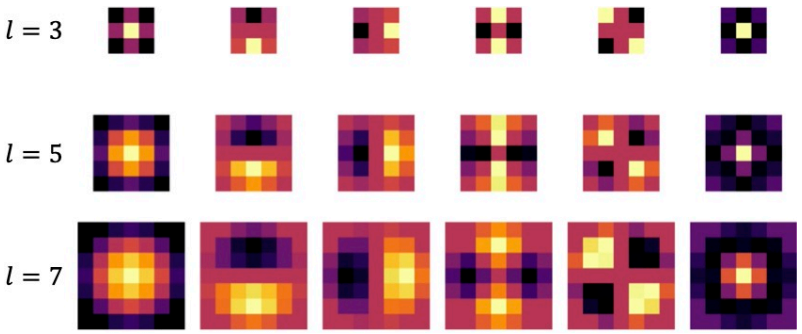


Figure B: Visualization of the first six Fourier-Bessel basis elements at three scales of $l = 3$, $l = 5$, and $l = 7$.

A.3. Convolutional Atom Generation Networks

To generate the dynamic atoms for per-pixel adaptive convolutional filters, we parametrize the generation network Φ as a light weight 2-layer convolutional neural network. The first layer uses 1×1 convolution to shrink the channel dimension of the input feature to 64 channels, and a 3×3 convolutional layer is then followed to output the prediction of dynamic atoms $\mathbf{D}$ or basis coefficients α, we consistently observe that this configuration delivers good results with very low cost, while increasing the size of Φ does not influence the performance significantly.

A.4. Network Structures of Ad-ResNets

Dynamic bottleneck blocks Following ResNet [11], we construct dynamic bottleneck blocks by using 1×1 convolutional layers to squeeze the restore the channel dimensions of feature maps, and applying ACDA on the intermediate features with fewer channels. This configuration fully demonstrates the advantage of ACDA to alleviate the demand for filters with high channel numbers. Therefore, networks with ACDA usually enjoy more compact model sizes with fewer parameters. We use ACDA with atom bases, which include three scales of Fourier-Bessel bases. As shown in Appendix Figure B, we use

Table A: Network architectures of Ad-ResNets. 'Conv, 3×3, 64' denotes standard 3×3 convolutional layers with 64 output channels. We use ACDA with Fourier-Bessel atom bases.

Layer name	Ad-ResNet-s	Ad-ResNet-m	Ad-ResNet-l
Conv0	Conv 7×7, 64, stride 2 Max pool 3×3, stride 2		
Conv1_x	[Conv, 3×3, 64 3×3, 64] ×2	[Conv, 1×1, 64 Conv, 3×3, 64 Conv, 1×1, 256] ×2	[Conv, 1×1, 64 Conv, 3×3, 64 Conv, 1×1, 256] ×2
Conv2_x	[Conv, 3×3, 128 3×3, 128] ×2	[Conv, 1×1, 128 Conv, 3×3, 128 Conv, 1×1, 512] ×2	[Conv, 1×1, 128 Conv, 3×3, 128 Conv, 1×1, 512] ×3
Conv3_x	[Conv, 1×1, 128 ACDA, 7×7, 128 Conv, 1×1, 256] ×2	[Conv, 1×1, 256 ACDA, 7×7, 256 Conv, 1×1, 512] ×2	[Conv, 1×1, 256 ACDA, 7×7, 256 Conv, 1×1, 512] ×8
Conv4_x	[Conv, 1×1, 256 ACDA, 5×5, 256 Conv, 1×1, 512] ×2	[Conv, 1×1, 512 ACDA, 5×5, 512 Conv, 1×1, 1024] ×2	[Conv, 1×1, 512 ACDA, 5×5, 512 Conv, 1×1, 1024] ×3
Output	Average pool 1000-d FC layer		

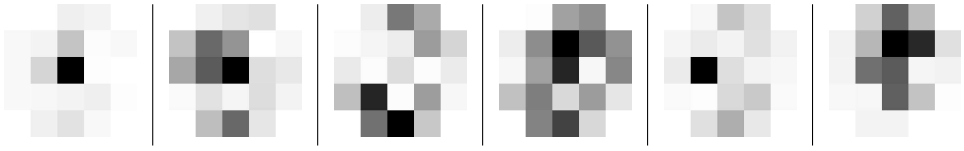


Figure C: We randomly select a layer and visualize the first predicted bases on a single ImageNet image at different spatial position. The bases are visually diverse.

$m' = 6$ and $S = 3$, i.e., three sets of Fourier-Bessel bases with 6 basis elements in each set. We empirically observe that this configuration with only 18 atom basis elements is sufficient to deliver satisfying performance while only requires a 18-dim vector to be predicted at each spatial position by the dynamic atom generation network to reconstruct a dynamic filter atom. At each layer, we adopt a tiny two-layer CNN as the dynamic atom generation network, whose architecture is detailed in Appendix Section A.3. In practice, we use 6 dynamic filter atoms, i.e., $m = 6$, so that the dynamic atom generation network output a $6 \times 18 = 108$-dim vector at each spatial position.

Ad-ResNet Without heavily tuning the network architectures, we follow ResNets and construct two variants of Ad-ResNet. The detailed network configurations are presented in Table A. In crowd counting experiments, the first 'Max pool' layer and the final 'Output' layer are removed. Consecutive transposed convolution layers with batch normalization and ReLU activation are used for resolution recovery.

A.5. Adaptivity and Diversity of Generated Atoms

To show that the proposed atom generator predicts atoms adaptively, we calculate the average pairwise cosine distance of basis coefficients $\boldsymbol{\alpha}$ (averaged across scales) within the spatial position of a single image from ImageNet. On the entire ImageNet validation set, we obtain 0.68 ± 0.30 intra-image cosine distance of $\boldsymbol{\alpha}$, which jointly with Figure C, show the diversity of the predicted bases (filters) both quantitatively and qualitatively.

A.6. Ablation Study

We present ablation study on *RealSR Final* $\times 3$ to validate the selection of the number of atoms m and the scales S of Fourier Bessel bases. All comparisons are in Table B. $m > 7$ and $S > 3$ lead to minor performance improvements only. For better trade-offs between performance and practical costs, we adopt $m = 6$ and $S = 3$.

Table B: Ablation study on number of atom m and scales S.

	Number of atoms m							Number of scales S				
	m = 3	m=5	m=7	m=9	m=11	m=13	m=15	S=1	S=2	S=3	S=4	S=5
PSNR	30.52	30.68	30.73	30.74	30.75	30.73	30.74	30.63	30.73	30.72	30.70	30.69
SSIM	0.858	0.866	0.868	0.868	0.869	0.866	0.868	0.866	0.868	0.868	0.865	0.866

B. Qualitative Results on Crowd Counting

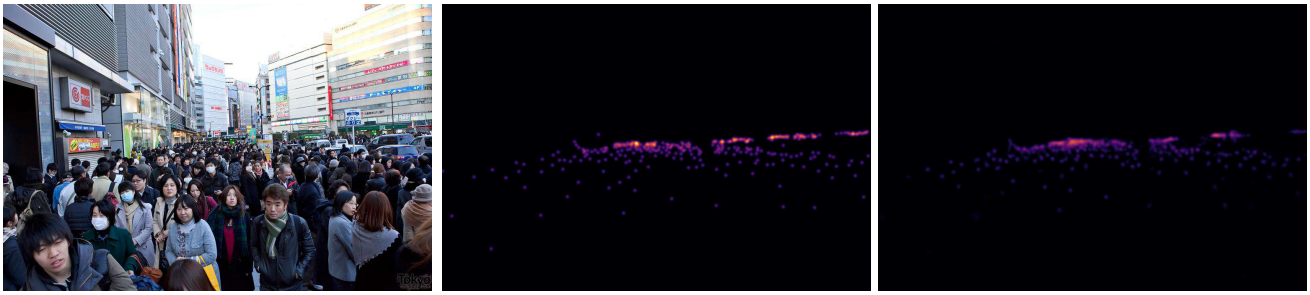
More qualitative results on crowd counting are presented in Figure D.

C. Qualitative Results on Super-resolution

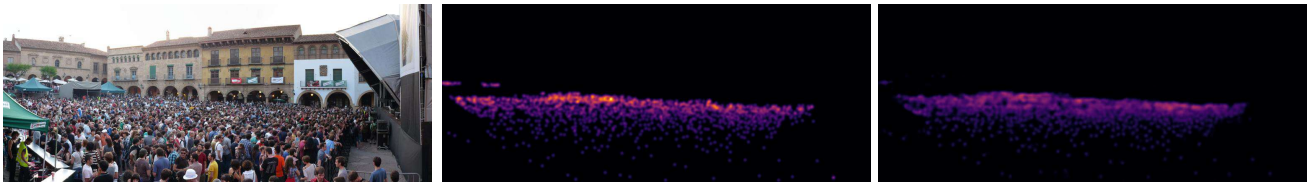
We present qualitative results for super-resolution experiments on RealSR dataset in Figure E.

D. Qualitative Results of Denoising

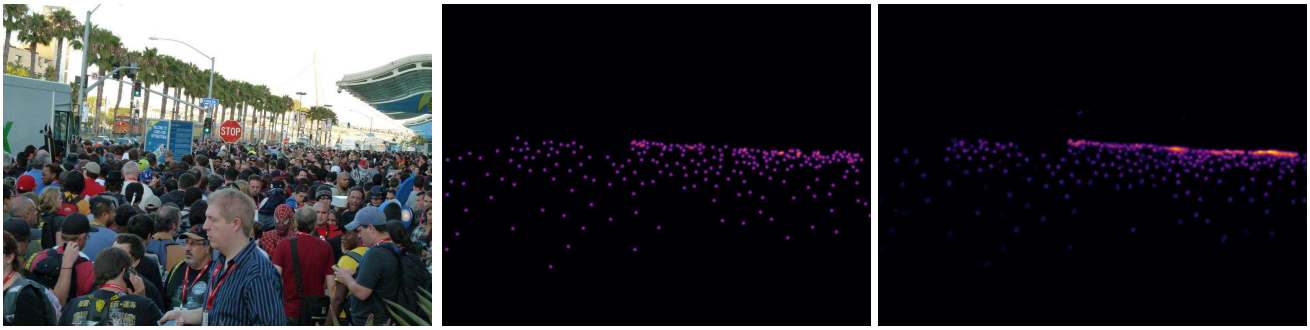
We present qualitative results and comparisons in Figure F and Figure G.



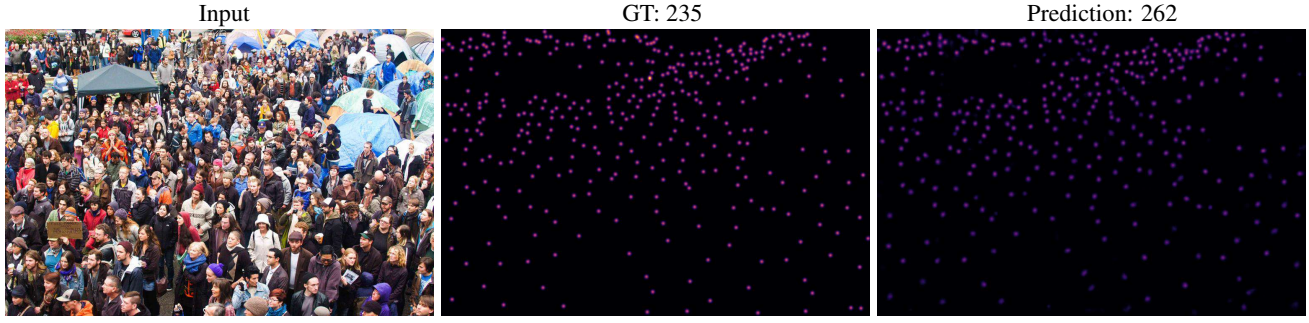
Input GT: 308 Prediction: 289



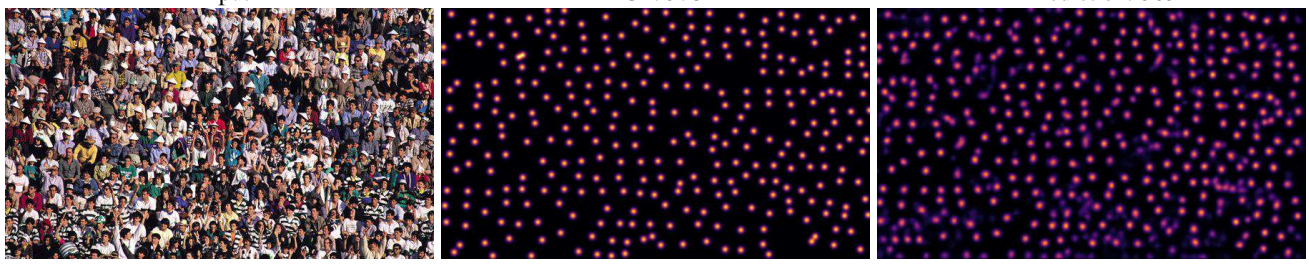
Input GT: 1265 Prediction: 1179



Input GT: 235 Prediction: 262



Input GT: 378 Prediction: 363



Input GT: 314 Prediction: 328

Figure D: Visualizations of the atom basis coefficients heatmaps. It is clearly shown that the adaptive convolutions tend to adopt large kernel sizes, i.e., with 7×7 atom bases, when the objects in the target regions have larger spatial sizes, i.e., the closer objects. While 3×3 bases are preferred when the dynamic convolutions are applied on regions with dense objects.

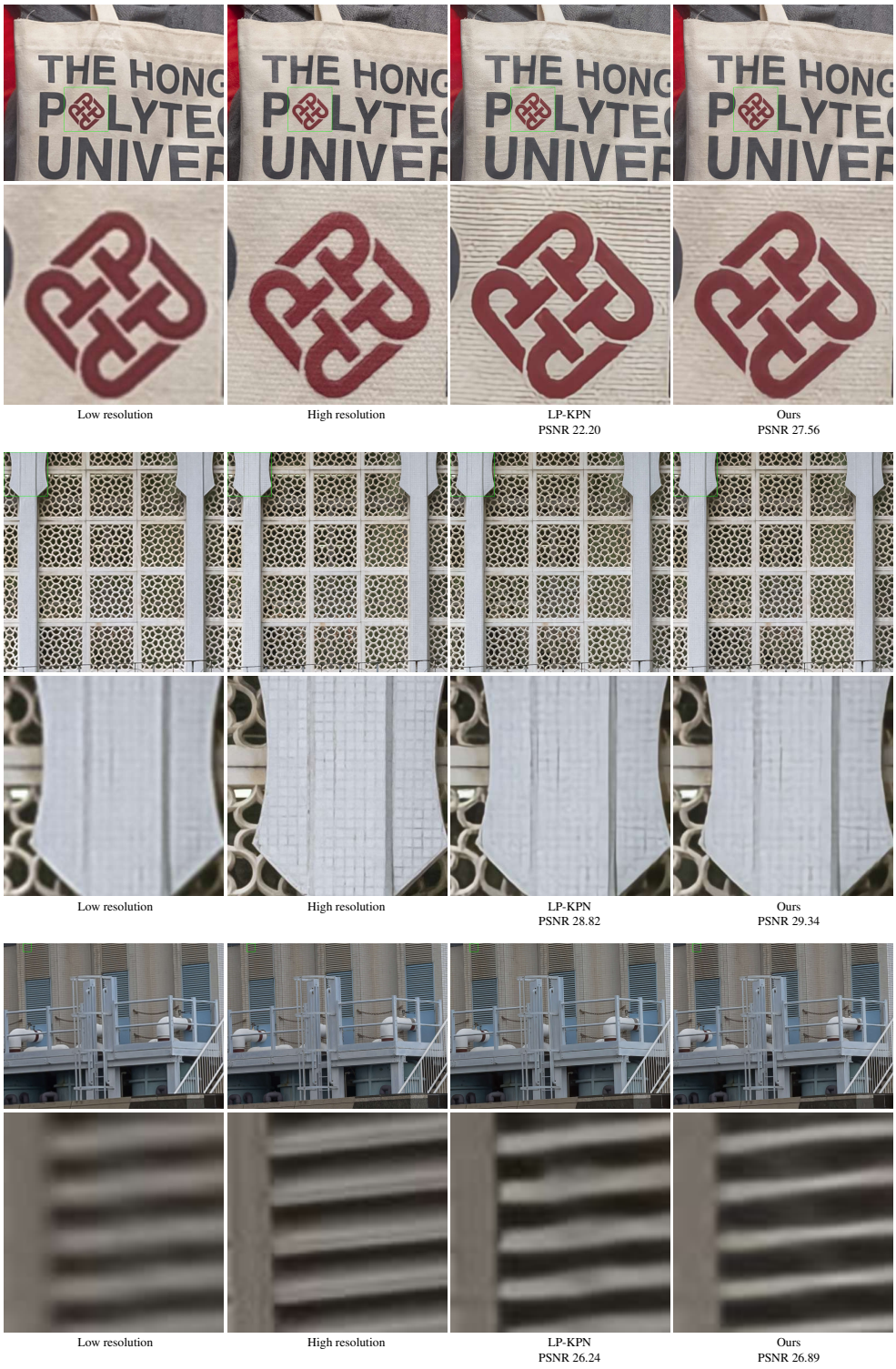


Figure E: Qualitative comparisons against LP-KPN. Although two methods deliver similar quantitative results, we consistently observe that LP-KPN suffers from strong artifacts. ACDA produce more faithful restoration results. We believe the adaptive convolutions operated in deep features help better capture image semantics, thus prevent over-sharp results with strong artifacts.

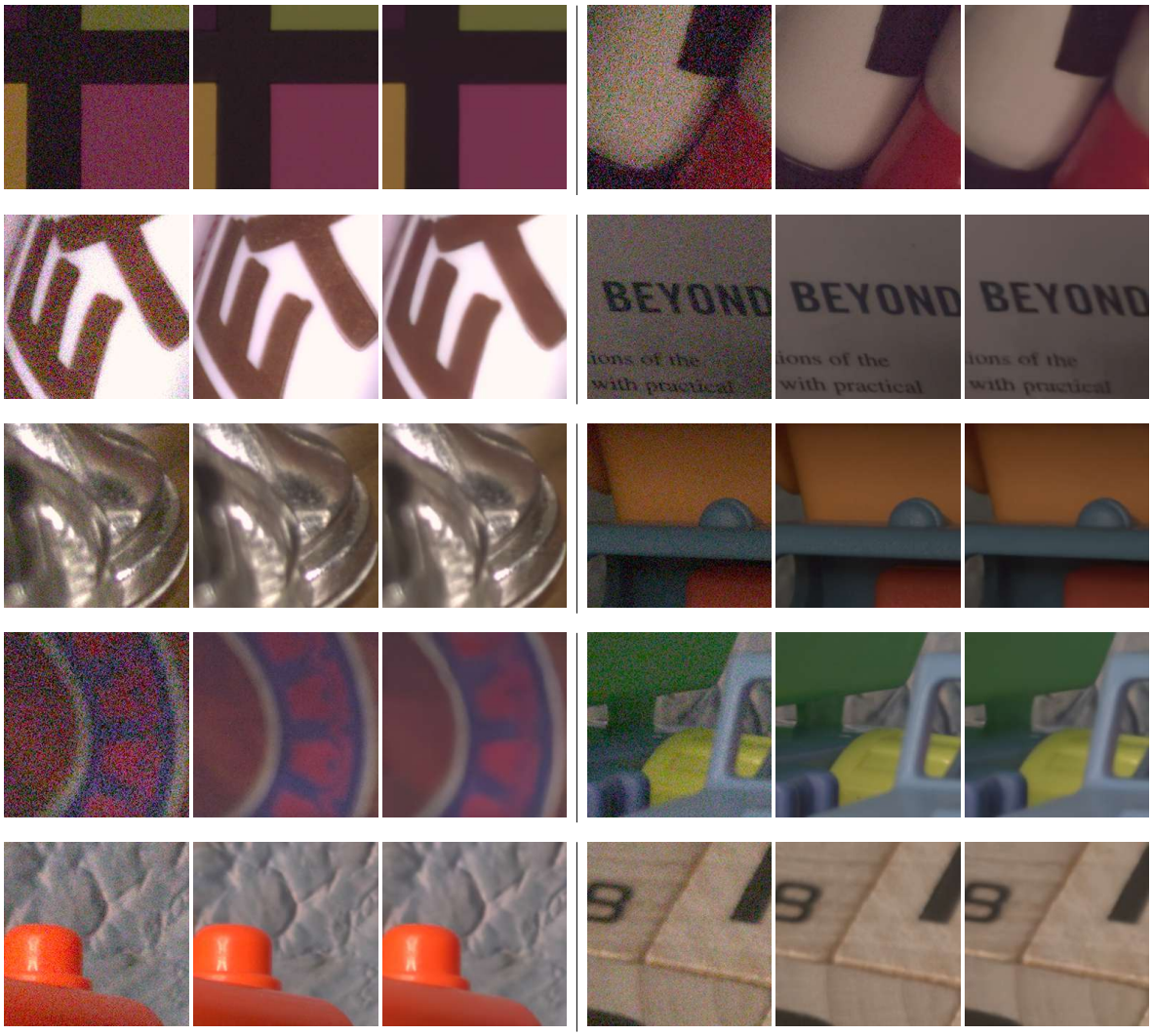
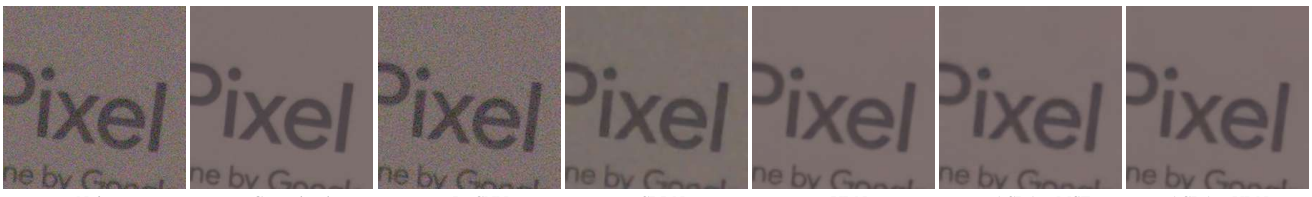


Figure F: Qualitative results on SIDD.

Noisy | Groundtruth | DnCNN PSNR 18.18 | CBDNet PSNR 21.13 | VDNet PSNR 32.64 | ACDA + MSE PSNR 32.13 | ACDA + VDNet PSNR 32.76



Noisy | Groundtruth | DnCNN PSNR 28.41 | CBDNet PSNR 35.18 | VDNet PSNR 40.82 | ACDA + MSE PSNR 40.22 | ACDA + VDNet PSNR 40.99

Figure G: Qualitative comparisons against multiple methods on SIDD.