# Fast and High Quality Image Denoising via Malleable Convolution

Yifan Jiang*[1], Bartlomiej Wronski[2], Ben Mildenhall[2], Jonathan T. Barron[2], Zhangyang Wang[1], and Tianfan Xue[2]

[1] University of Texas at Austin
[2] Google Research

**Abstract.** Most image denoising networks apply a single set of static convolutional kernels across the entire input image. This is sub-optimal for natural images, as they often consist of heterogeneous visual patterns. Dynamic convolution tries to address this issue by using per-pixel convolution kernels, but this greatly increases computational cost. In this work, we present **Malle**able **Conv**olution (**MalleConv**), which performs spatial-varying processing with minimal computational overhead. MalleConv uses a smaller set of spatially-varying convolution kernels, a compromise between static and per-pixel convolution kernels. These spatially-varying kernels are produced by an efficient predictor network running on a downsampled input, making them much more efficient to compute than per-pixel kernels produced by a full-resolution image, and also enlarging the network's receptive field compared with static kernels. These kernels are then jointly upsampled and applied to a full-resolution feature map through an efficient on-the-fly slicing operator with minimum memory overhead. To demonstrate the effectiveness of MalleConv, we use it to build an efficient denoising network we call **MalleNet**. MalleNet achieves high-quality results without very deep architectures, making it 8.9× faster than the best performing denoising algorithms while achieving similar visual quality. We also show that a single MalleConv layer added to a standard convolution-based backbone can significantly reduce the computational cost or boost image quality at a similar cost. More information are on our project page: `https://yifanjiang.net/MalleConv.html`

**Keywords:** Image Denoising, Dynamic Kernel, Efficiency

## 1 Introduction

Image denoising is a fundamental problem to computational photography and computer vision. Recent advances in deep learning have sparked significant interest in learning an end-to-end mapping directly from corrupted observations to the unobserved clean signal, without an explicit model of signal corruptions. These networks appear to learn a prior over the appearance of "ground truth"

* This work was performed while Yifan Jiang worked at Google.

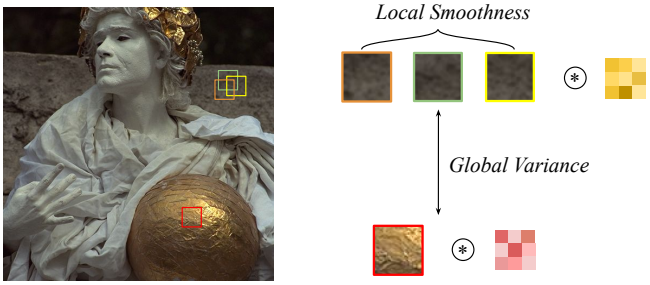


**Fig. 1. Local smoothness and global variance in natural images.** Our proposed MalleConv layer applies spatially-varying filters for features in different contexts and adopt similar filters in areas that are locally smooth, thus balancing the trade-off between global variance and local smoothness.

noise-free images in addition to the statistical properties of the noise present in the inputs.

The performance of denoising networks has consistently been improved with deeper and wider layers, as they can extract richer representations and also increase the receptive field. However, deeper and wider layers also significantly amplify computational costs and the difficulty of optimization. One hurdle is that most of neural architectures only apply a single fixed set of convolutional kernels over the entire input, exploiting spatial equivariance for computational efficiency. However, natural images often contain spatially heterogeneous visual patterns, depriving the convolution of the ability to adapt to globally varying features.

One recent effort addresses this issue is a kernel prediction network (or "hypernetwork") [5,25,29,45,57,61], which generates spatially-varying kernels at each pixel location. Although applying per-pixel kernels increases representational power, it also greatly increases computational cost, as the number of kernels grows with the image resolution. This makes it particularly challenging for mobile cellphone cameras, which normally have about 12 megapixels, and very limited compute resources and power budget.

To achieve spatial-varying processing while maintaining low computational cost, we propose an efficient variant of spatially-varying kernels, dubbed Malleable Convolution (**MalleConv**). We draw inspiration from the trade-off between local smoothness and global spatial heterogeneity. Fundamentally, natural images contain spatially-varying patterns from a "global" perspective, which motivates the popularity of dynamic filters [25,29] and self-attention modules [38,56], but image content only changes slowly in a "local" neighborhood. Therefore, natural image patches tend to redundantly recur many times inside the image, both within the same scale and across different scales [48,20]. Natural image textures are also commonly represented as a fractal set with self-similarity at all scales [32]. Examples in Fig. 1 also illustrate this phenomenon. The golden ball held by the man contains different patterns compared to the stone in the background,

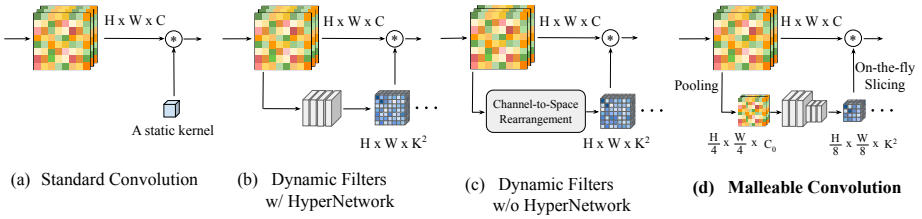


**Fig. 2. Comparing MalleConv with static filter and other dynamic filters.** (a) Standard convolution with a static kernel. (b) Generate dynamic filters using a HyperNetwork [25,29]. (c) Generate dynamic filters using a channel-to-space operation [36]. (d) Our Malleable convolution.

but the texture is locally consistent within a region of stone. Therefore, those similar content can share the same set of kernels to save compute.

Based on this observation, we proposed MalleConv, which scales per-pixel dynamic filter approach to a larger region. Specifically, unlike dynamic filters which take full-resolution input and generate full-resolution kernels, MalleConv only processes a downsampled representation, outputting location-specific dynamic filters at **a much smaller spatial resolution** compared with the original feature map (Fig. 2(d)). These kernels are later applied to the full-resolution feature map using a "slicing" strategy, which fuses on-the-fly bilinear interpolation and convolution into a single operator. This design has several advantages. First, comparing to the hypernetwork used in dynamic filters, our predictor network only takes a low-resolution feature map as input to keep it light-weight. Second, full resolution per-pixel kernels are calculated and applied in the same operation, without requiring additional memory I/O for storing and retrieving the high resolution kernel map. Together, these significantly reduce computational overhead compared to full-resolution dynamic filters. Moreover, by taking a downsampled image as input, the predictor network has a large receptive field without very deep structure.

Comprehensive experiments are conducted to demonstrate the effectiveness of the proposed method. We evaluate MalleNet on public synthetic and real image benchmarks (Synthetic: CBSD68, Kodak24, McMaster; Real: SIDD and DND). In addition, we conduct ablation study by injecting MalleConv into existing backbones, including DnCNN [67], UNet, and RDN [72], where the results show that MalleConv achives better quality-efficiency trade-off compared to other dynamic kernels.

In summary, our contributions are as follows:

- We propose Malleable Convolution (MalleConv), a new spatially-varying kernel layer that serves as a powerful variant of standard convolution. MalleConv largely benefits from an efficient predictor network, which incurs minimum additional cost to achieve a spatial-varying processing.

- We conduct a comprehensive ablation study by inserting MalleConv into various popular backbone architectures (including DnCNN, UNet, and RDN), where we show MalleConv can reduce runtime by up to **20**× with similar visual quality.
- We compare MalleConv with previous spatially-varying kernel architectures including HyperNetworks [25] and Involution [36]. MalleConv demonstrates a better quality-efficiency trade-off.
- We further design a new MalleNet architecture using the proposed MalleConv block, achieving faster performance and higher quality on both synthetic and real-world denoising benchmarks.

## 2 Related Work

### 2.1 Image Denoising

Traditional image denoising algorithms make use of information in local pixel neighborhoods [47,51] or sparse image prior [3,16,43,6,18,13]. Recently, deep convolutional networks have demonstrated success in many image restoration tasks [15,40,72,54,34,35,50,37,62,30,59,11,31]. For image denoising specifically, Burger et al. [7] proposed a plain multi-layer perception model that achieves comparable performance to BM3D. Chen et al. [10] proposed a trainable nonlinear reaction diffusion model that learns to remove additive white gaussian noise (AWGN) by unfolding a fixed number of inference steps. Many subsequent works further improved upon it by using more elaborate neural network architecture designs, including residual learning [67], dense networks [72], non-local modules [71,8,38], dilated convolutions [46], and more [12,65,64,9]. However, many of these approaches use heavy network architectures that are often impractical for mobile use cases. To tackle this issue, several recent works focus on fast image denoising, by either introducing a self-guidance network [22] or increasing the nonlinear model capacity [21]. In contrast, our approach relies on spatially-varying kernels, where parameters are dynamically generated by an efficient prediction network.

### 2.2 Dynamic Filters and Spatially Varying Kernels

Convolutional neural networks producing dynamic kernels have been widely studied for a variety of applications. The pioneering works [29,25] adopt a parameter-generating network to produce location-specific filters. These works directly produce spatially-varying weights for the whole convolutional layer, substantially increasing the latency and computational cost of their approaches. Wang et al. [57] designed a feature upsampling module (CARAFE) that generates kernels and reassembling features inside a predefined nearby region. However, CARAFE is designed as a feature upsampling operator instead of a variant of convolution. The context-gated convolution [41,73] adopts a gated module and channel/spatial interaction module to generate modified convolutional kernels. Although their

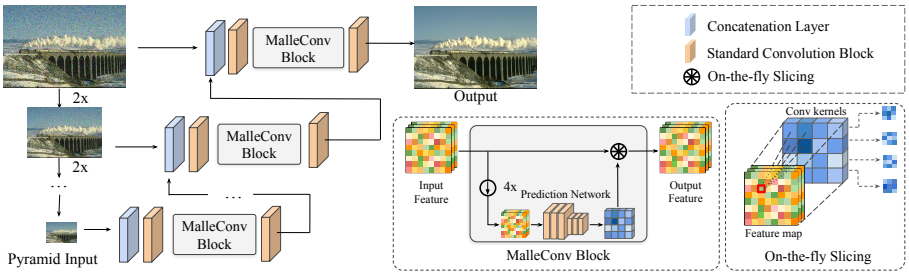


**Fig. 3. Main architecture of MalleNet.** MalleNet takes a 4-level image pyramid as input. Each layer consists of several Inverted Bottleneck Blocks with a MalleConv block inserted in between. Bottom middle shows the structure of MalleConv block, which consists of a small prediction network and a on-the-fly slicing operator. Bottom right shows details of on-the-fly slicing operator. For each input feature (red rectangle), four neighboring kernels are bilinearly combined and applied to that feature to generate the corresponding output feature.

filter weights are produced dynamically, they apply the same filter at different spatial locations. Another line of work [36] avoids using a hypernetwork by employing a channel-to-space rearrangement to generate location-specific filters. Without the help of a hypernetwork, this approach can not capture the local information and image context. While previously described approaches mainly adopt dynamic filters inside multiple convolutional layers of a deep network, a different line of work[5] proposed to use a standard convolutional neural network to predict denoising kernels that are applied directly to the input to produce the target image. Mildenhall et al. [45] extended this approach to burst denoising by predicting a separate set of weights for each image in a temporal sequence. HDRNet [19] uses a deep neural network to process the low-resolution input and applies the produced spatially-varying affine matrix to the full-resolution input by slicing a predicted bilateral grid. In stead of processing the input image, our proposed Malleable Convolution applies an efficient predictor network to process a downsampled feature map, then constructs a deep spatially-varying network layer-by-layer.

## 3 Method

### 3.1 Preliminaries

A standard convolutional layer applies a kernel with weights $W \in \mathbb{R}^{C_{in} \times C_{out} \times K^2}$ to an input feature map sampled from a 2D tensor $X \in \mathbb{R}^{C_{in} \times H \times W}$. Here $H, W$ are the height and width of the feature map, $C_{in}, C_{out}$ denote the numbers of input and output channels, and $K$ is the kernel size. This basic design struggles to capture global context information and cannot adapt to different regions of natural images that contain spatially heterogeneous patterns. Although previous works address this issue by adopting per-pixel dynamic filters [25,29,45] or

generating spatial-agnostic filters via a channel-to-space permutation [36], their approaches either require large memory footprint or do not capture context information.

### 3.2 Malleable Convolution with Efficient Predictor Network

To overcome the aforementioned drawbacks, we propose a new operation, dubbed Malleable Convolution (MalleConv). MalleConv is equipped with an light-weight predictor network that significantly reduces the memory cost and runtime latency of previous dynamic kernel prediction [25,29,45]. The proposed predictor network first downsamples the input feature map $X$ to $X^{'} \in \mathbb{R}^{\frac{H}{4} \times \frac{W}{4} \times C}$ through a $4 \times 4$ average pooling. After that, we build a light-weight predictor network consists of multiple ResNet blocks [27] and max pooling layers [26] (see supplementary materials for detailed architecture). The predictor network outputs a feature map $Y \in \mathbb{R}^{\frac{H}{8} \times \frac{W}{8} \times C^{'}}$, where $C^{'} = K^2 \times C$. To formulate a spatially-varying filter, the learned representation $Y$ is reshaped to a list of filters $\{W_{ij}\} \in \mathbb{R}^{K^2 \times C}$, where $i \in \{1, 2, ..., \frac{H}{8}\}$, $j \in \{1, 2, ..., \frac{W}{8}\}$. Each kernel in $Y$ only has $C$ channels, not $C_{in} \times C_{out}$, as we use depth-wise convolution [28] to further reduce the number of parameters. Finally, we upsample the learned spatially-varying filters $\{W_{ij}\}$ through bilinear interpolation to obtain per-pixel filters $\{W_{ij}^{'}\} \in \mathbb{R}^{K^2 \times C}$, where $i \in \{1, 2, ..., H\}$, $j \in \{1, 2, ..., W\}$, and independently apply them to the corresponding input channels.

### 3.3 Efficient On-the-fly Slicing

A naive way to implement malleable convolution is to first upsample the low-resolution filters to full-resolution using bilinear interpolation and then apply them to the full-resolution feature map. However, this introduces a large memory footprint since the high-resolution kernels are being precomputed and stored before their application.

To mitigate the memory issue, we combine these two steps into a on-the-fly slicing operator. It takes in a high-resolution feature map $X \in \mathbb{R}^{H \times W \times C}$ and low-resolution kernel maps $\{W_{ij}\} \in \mathbb{R}^{K^2 \times C}$ as input. The result of the on-the-fly slicing operator is a new feature map $Z$ with the same resolution as $X$. For each pixel location, we first calculate the bilinear interpolated kernel weights from four neighboring kernels as (also illustrated in bottom right of Fig. 3)

$$W_{x,y}^{'} = \sum_{i,j \in N(x,y)} \tau(r_x x - i)\tau(r_y y - j)W'_{i,j}, \tag{1}$$

where $\tau$ is the linear interpolation operator $\tau(a) = max(1 - |a|, 0)$, $r_x$ and $r_y$ are the width and height ratios of the low-resolution filters w.r.t. the full resolution input feature map, and $N(x, y)$ is the four-neighborhood. Bias term $b_{x,y}^{'}$ is sliced in the similar way. The output feature $Z$ is then calculated as:

$$Z_{x,y}(c) = W_{x,y}^{'}(c) \cdot X_{x,y}(c) + b_{x,y}^{'}(c), \tag{2}$$

| Method | Latency/(ms) | Flops/(G) | CBSD68 | | | Kodak24 | | | McMaster | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | $\sigma = 15$ | $\sigma = 25$ | $\sigma = 50$ | $\sigma = 15$ | $\sigma = 25$ | $\sigma = 50$ | $\sigma = 15$ | $\sigma = 25$ | $\sigma = 50$ |
| BM3D [13] | 41.56 | - | 33.52 | 30.71 | 27.38 | 34.28 | 32.15 | 28.46 | 34.06 | 31.66 | 28.51 |
| FFDNet [69] | - | 7.95 | 33.87 | 31.21 | 27.96 | 34.63 | 32.13 | 28.98 | 34.66 | 32.35 | 29.18 |
| **MalleNet-S** | **4.62** | **2.93** | **33.90** | **33.22** | **27.97** | **34.66** | **32.16** | **29.00** | **34.68** | **32.35** | **29.20** |
| RPCNN [60] | 95.11 | - | - | 31.24 | 28.06 | - | 32.34 | 29.25 | - | 32.33 | 29.33 |
| DSNet [46] | - | - | 33.91 | 31.28 | 28.05 | 34.63 | 32.16 | 29.05 | 34.67 | 32.40 | 29.28 |
| IRCNN [68] | - | 12.18 | 33.86 | 31.16 | 27.86 | 34.69 | 32.18 | 28.93 | 34.58 | 32.18 | 28.91 |
| DnCNN [67] | 21.69 | 68.15 | 33.90 | 31.24 | 27.95 | 34.60 | 32.14 | 28.95 | 33.45 | 31.52 | 28.62 |
| DnCNN* [67] | 21.69 | 68.15 | 34.02 | 31.34 | 28.11 | 34.62 | 32.18 | 29.11 | 35.18 | 32.73 | 29.49 |
| **MalleNet-M** | **16.69** | **9.36** | **34.15** | **31.50** | **28.27** | **34.82** | **32.41** | **29.35** | **35.53** | **33.12** | **29.96** |
| BRDNet [55] | - | - | 34.10 | 31.43 | 28.16 | 34.88 | 32.41 | 29.22 | 35.08 | 32.75 | 29.52 |
| DRUNet [66] | - | 102.91 | 34.30 | 31.69 | 28.51 | **35.31** | **32.89** | **29.86** | 35.40 | 33.14 | 30.08 |
| **MalleNet-L** | **32.34** | **33.47** | **34.32** | **31.71** | **28.52** | 34.93 | 32.58 | 29.50 | **35.65** | **33.26** | **30.12** |
| RNAN [71] | - | 774.67 | - | - | 28.27 | - | - | 29.58 | - | - | 29.72 |
| RDN [72] | 263.03 | 2001.86 | - | - | 28.31 | - | - | 29.66 | - | - | - |
| RDN* [72] | 263.03 | 2001.86 | 34.29 | 31.69 | 28.37 | 34.89 | 32.52 | 29.68 | 35.55 | 33.16 | 29.92 |
| IPT [8] | - | 938.66 | - | - | 28.39 | - | - | 29.64 | - | - | 29.98 |
| SwinIR [38] | 780.61 | 788.10 | 34.42 | 31.78 | 28.56 | **35.34** | **32.89** | **29.79** | 35.61 | 33/20 | 30.22 |
| **MalleNet-XL** | **87.55** | **181.89** | **34.54** | **31.86** | **28.62** | 35.07 | 32.67 | 29.61 | **35.72** | **33.28** | **30.23** |

**Table 1. Comparing MalleNet with the state-of-the-art methods on three common benchmarks.** We try our best to use the official implementation provided by the authors to calculate FLOPs and latency. "*" denotes that the original methods were trained with small-scale dataset and we retrain these networks with more training data and larger patch size, for fair comparison.

where $c$ is the channel index. Note that the sliced weight $W^{'}$ and bias $b^{'}$ are calculate on-the-fly without additional memory cost. We discuss more about the specific memory consumption in Sec. 4.4.

### 3.4 Malleable Network

As the goal of this work is to design an ultra-fast denoiser, current state-of-the-art algorithms such as the residual dense network [72] or transformer-based architectures [8,38] are sub-optimal to build an efficient backbone. Inspired by some recent pyramid-based approaches [22,39,64], we design a new backbone integrating the proposed malleable convolution, dubbed **MalleNet**.

MalleNet first builds a four-level pyramid using $2\times$ space-to-channel shuffle operations [53]. This allows us to extract multi-scale representations and increases the network's receptive field. In each stage, we stack several Inverted Bottleneck Blocks [52] with a fixed ratio and insert one $K \times K$ Malleable Convolution in-between to extract heterogeneous representations. At the end of the bottom stage, we upsample the feature map and concatenate it with the input of its upper stage. In the top stage, the representation extracted from different pyramids are aggregated to produce the final output. Compared to conventional encoder-decoder style architectures, the pyramid-based architecture reuses the extracted representation from each scale and thus can achieve faster inference speed. The whole network is shown in Fig. 3.

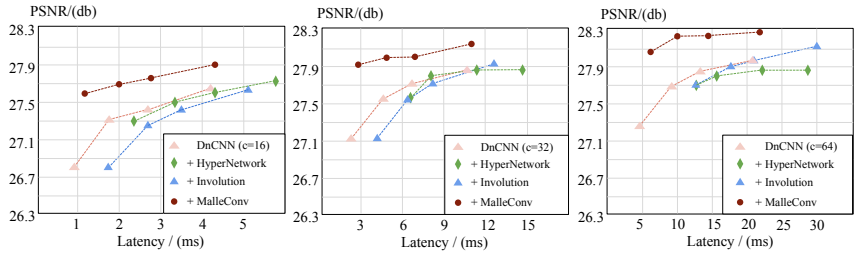


**Fig. 4.** Comparison between MalleConv and other dynamic filters in terms of runtime latency and PSNR value.

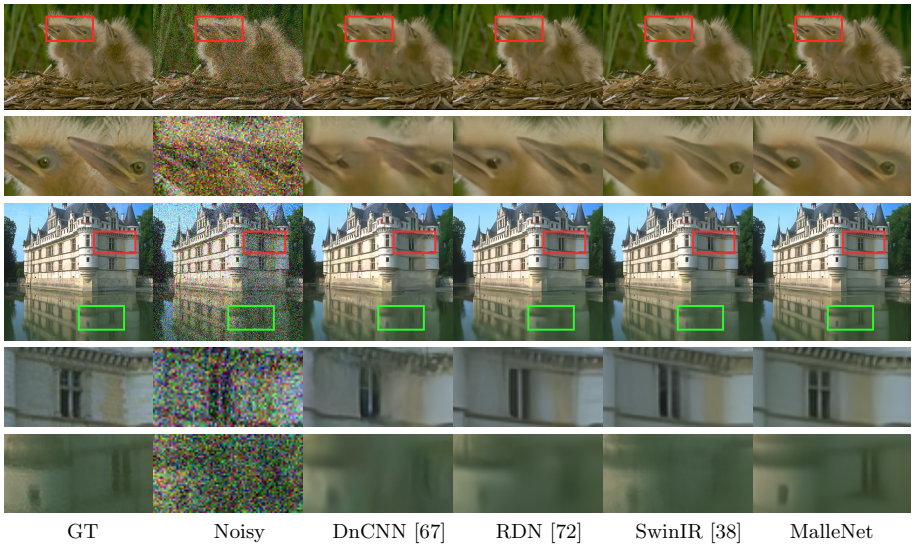


**Fig. 5.** Visual comparison between MalleNet and previous approaches. More visual results are shown in the supplementary.

## 4 Experiments

We mainly evaluate the proposed module on the Additive White Gaussian Noise (AWGN) removal task. Following previous work [66], we construct a training dataset with 400 examples from the Berkeley Segmentation Dataset (BSD) [44], 4,744 examples from the Waterloo Exploration Database [42], 900 images from the DIV2K dataset [2], and 2,750 images from the Flick2K dataset [40]. We adopt $160 \times 160$ training patch size, which we augment through random cropping, rotations, and flipping. Other networks (e.g., IPT [8] and SwinIR [38]) are not able to be benefited from larger patch size, due to the heavy memory cost. We empirically choose kernels size $1 \times 1$ for MalleConv on AWGN removal tasks and kernel size $3 \times 3$ on real-world benchmarks, as that is observed to reach the best PSNR-to-Complexity trade-off. We adopt the Adam optimizer [33] with a batch

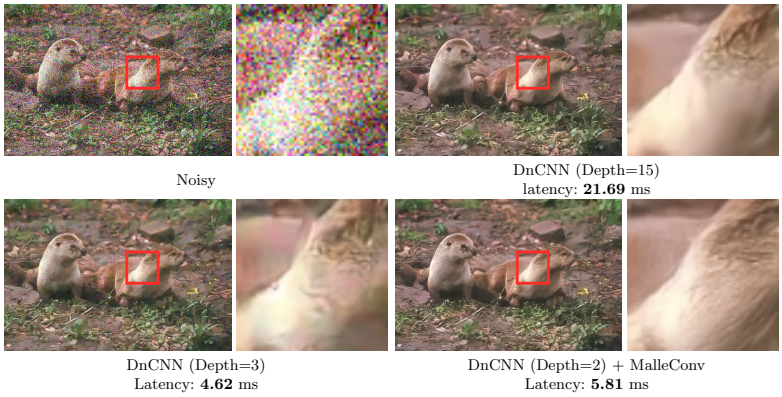


**Fig. 6.** Visual results by inserting MalleConv into a fast variant of DnCNN, with $\sigma = 50$.

size of 16 and a cosine learning rate scheduler. The initial learning rate is set to 0.001. The full training process takes 2.2M iterations. We adopt 3 common datasets as our testing set: CBSD68, kodak24 [17], and McMaster [70].

All of our experiments are conducted on 8 Nvidia V100 GPUs using the Tensorflow-2.6 platform. The FLOPs (floating point operations) and runtime are calculated on a $256 \times 256$ resolution RGB patches. We benchmar the inference speed on a single Nvidia P6000 GPU platform by setting batch size set to the maximum available number. For PyTorch-based implementations, we report the average latency of a single $256 \times 256 \times 3$ input collected from 500 runs. For Tensorflow-based implementations, we report the latency time using the Tensorflow official profiler[3].

### 4.1 Comparing MalleConv with Other Dyanmic Kernels

To demonstrate the efficiency and effectiveness of the proposed MalleConv, we compare specific computational cost and performance of each individual network equipped with MalleConv and other dynamic filters, e.g., HyperNetwork [25] and Involution [36]. We adopt DnCNN [67] as our main backbone and replace the middle layer of DnCNN with a single dynamic filter operator. We evaluate three different DnCNN backbones with channel=$\{16, 32, 64\}$. In each one, the number (depth) of DnCNN backbone are growing from 3, 6, 9, to 15. As shown in Fig. 4, MalleConv achieves the best performance-efficiency trade-off by significantly improving the PSNR with minimum additional runtime latency.

### 4.2 Comparing with State-of-the-Art Methods

To fairly compare the runtime speed between MalleNet and other baselines, we train 4 versions of MalleNet: -S, -M, -L, and -XL by increasing the number

[3] https://www.tensorflow.org/guide/profiler

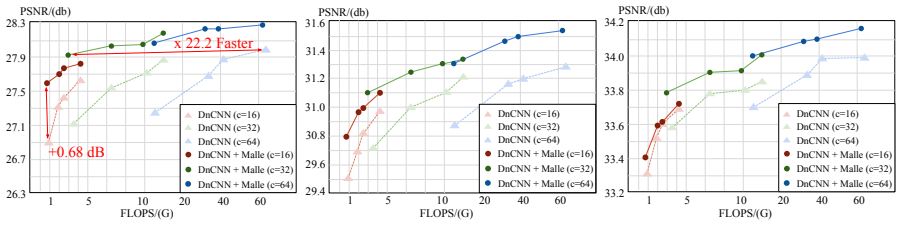


**Fig. 7. PSNR-To-Complexity trade-off of DnCNN and DnCNN with a single MalleConv**. We build DnCNN-families by setting depth = {3, 6, 9, 15} and channel = {16, 32, 64}. The three figures from left to right show experiments with $\sigma$ = {50, 25, 15}.

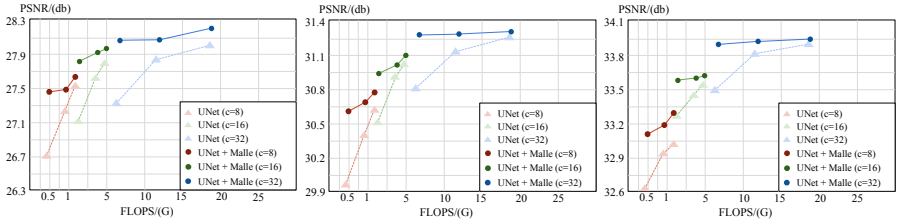


**Fig. 8. PSNR-To-Complexity trade-off of UNet and UNet with a single MalleConv**. We build UNet-families by setting the encoder-decoder block number = {2, 3, 4} and channel = {8, 16, 32}. The three figures show experiments with $\sigma$ = {50, 25, 15}.

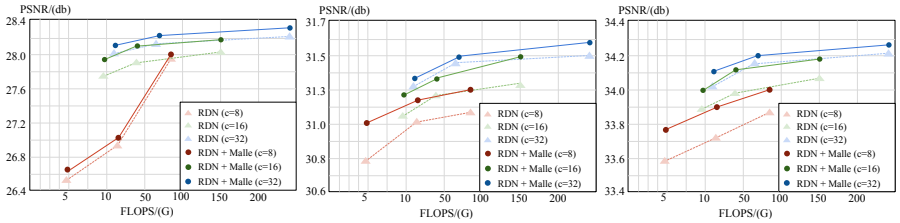


**Fig. 9. PSNR-To-Complexity trade-off of RDN and RDN with a single MalleConv**. We build RDN-families by setting the residual dense block number = {3, 6, 10} and channel = {8, 16, 32}. The three figures show experiments with $\sigma$ = {50, 25, 15}.

| Depth | Metrics | AvgPooling Size | | | |
|---|---|---|---|---|---|
| | | 0 | 2 | 4 | 8 |
| D=3 | Latency/(ms) | 13.19 | 7.08 | 5.62 | 5.18 |
| | FLOPs/(G) | 43.96 | 18.17 | 11.71 | 10.10 |
| | PSNR/(dB) | 27.91 | **28.15** | 28.07 | 28.01 |
| D=6 | Latency/(ms) | 17.39 | 11.28 | 9.85 | 9.43 |
| | FLOPs/(G) | 62.05 | 36.26 | 29.81 | 28.19 |
| | PSNR/(dB) | 28.19 | **28.24** | **28.24** | 28.18 |
| D=15 | Latency/(ms) | 30.09 | 23.98 | 22.53 | 22.09 |
| | FLOPs/(G) | 98.24 | 72.44 | 66.00 | 64.39 |
| | PSNR/(dB) | 28.25 | 28.28 | **28.31** | 28.28 |

**Table 2.** Ablation study on the size of AvgPooling layer in MalleConv Operator. PSNR results are reported on the CBSD68 test set with $\sigma = 50$.

of channels from 16, 32, 64 to 144. We divide evaluated approaches into four categories according to their performance and runtime speed. As shown in Table 1, on these four categories, MalleNet achieves the best efficiency-performance trade-off and reaches state-of-the-art results among two of our main benchmark test sets. We show the PSNR-to-Complexity trade-off of each method in Fig. 12 left.

### 4.3 MalleConv Layer with Alternative Backbones

To further demonstrate that the proposed Malleable Convolution can benefit wide variety of network architectures, we perform ablation studies by inserting MalleConv into existing well-known backbones as a plug-in operator. Here we choose three popular backbones as our main testbeds. Since most of original network structures are too heavy for edge devices, we also manually build a few cheaper variants by controlling the depth and channel variables. Using DnCNN as an example, the vanilla DnCNN architecture contains 15 layers with 64 channels. We construct its faster version by setting the depth = {3, 6, 9, 15} and channel = {16, 32, 64}, respectively, and obtain the architecture series of DnCNN with $3 \times 4 = 12$ variants.

Afterwards, we construct a number of better performing variants of these architecture series, **by replacing one standard convolution with a single $1 \times 1$ MalleConv operator**. We replace the middle layer of the network with a MalleConv block (detailed architectures are shown in the supplementary material). We conduct experiments on CBSD69 dataset and train these architectures using the same training recipes. As shown in Fig. 7, 8, and 9, a single MalleConv block brings significant improvement to all three backbones.

### 4.4 Visual Comparison and Interpretation

We first compare our best architecture MalleNet-XL with previous state-of-the-art approaches [67,72,38,38], as shown in Fig. 5. The examples produced by MalleNet preserve rich details and impressive textures while saving up to $\times$**8.91**

| Method | Latency | FLOPs/(G) | SIDD | | DND | |
|---|---|---|---|---|---|---|
| | | | PSNR | SSIM | PSNR | SSIM |
| DnCNN [67] | 21.69 | 68.15 | 23.66 | 0.583 | 32.43 | 0.79 |
| BM3D [13] | 41.56 | - | 25.78 | 0.685 | 34.51 | 0.851 |
| WNNM [23] | - | - | 25.78 | 0.809 | 34.67 | 0.865 |
| CBDNet [24] | - | - | 30.78 | 0.754 | 38.06 | 0.942 |
| RIDNet [4] | 98.13 | - | 38.71 | 0.914 | 39.26 | 0.953 |
| VDN [63] | - | - | 39.28 | 0.909 | 39.38 | 0.952 |
| ACDA [58] | - | - | 39.32 | 0.912 | - | - |
| MPRNet [65] | - | 573.50 | 39.71 | 0.958 | 39.80 | 0.954 |
| NBNet [12] | 37.44 | 88.70 | 39.75 | 0.973 | 39.89 | 0.955 |
| MIRNet [64] | 192.61 | 787.04 | 39.72 | 0.959 | 39.88 | 0.956 |
| HINet [9] | 32.83 | 170.71 | 39.99 | 0.958 | - | - |
| **MalleNet-R** | **13.58** | **29.11** | 39.56 | 0.941 | 39.21 | 0.949 |

**Table 3. Comparing MalleNet with the State-of-the-art methods on real-world benchmark SIDD and DND.** We try our best to use the official implementation provided by the authors to calculate the FLOPs cost and runtime speed.

inference time compared to the best baseline, further demonstrating the effectiveness of our approach. Moreover, in the "ultra-fast" setting, we decrease the depth of DnCNN from 15 to 3 to obtain a much faster variant of DnCNN architecture. However, the image quality also degrades as shown in the bottom-left of Fig. 6. In contrast, when replacing the middle layer of DnCNN with a single $1 \times 1$ MalleConv operator (DnCNN w/ MalleConv), it uses slightly more computational time, but achieves significantly better visual quality, as shown in the bottom-right of Fig. 6.

Furthermore, to illustrate how spatially-varying kernels in MalleConv capture heterogeneous visual patterns, we replace the spatially varying kernels in MalleNet with one selected kernel and apply it to the entire image. Fig. 11 compares the default output of MalleNet (column 2) with the one that applies a selected kernel (columns 3 and 4). When a kernel generated from a sky region (column 3) is applied, the network is observed to denoise the rest of the image as if they are the sky. Similarly, using a kernel from a snowy-mountain patch will generate output that looks like snowy mountain (column 4). By combining kernels that are dedicated to different local image statistics together, MalleConv can better model the heterogeneous spatial patterns and yield better results.

### 4.5 Analysis of Runtime Latency and Memory Cost

In Fig. 10, we compare the memory cost of each operator during the training process. We conduct our testbed on three different modules: 1) The $1 \times 1$ MalleConv with input and output channel to be 16, 32, 64. MalleConv generates smaller-size of dynamic kernels and then applies it back to the full-resolution feature using on-the-fly slicing operator; 2) We directly upsample the generated dynamic kernel via an $8\times$ bilinear upsampling operator, to match the resolution of input features; 3) We remove the downsampling and maxpooling layers in the

proposed efficient prediction network, thus it will generate per-pixel dynamic kernels and apply it to the feature map. As shown in Fig. 10, the memory cost of MalleConv is much smaller than other two counterparts, since it only needs to predict a smaller-size of filters compared to the per-pixel kernel prediction methods, and does not store the intermediate feature map of upsampled kernels compared to the bilinear interpolation operator.

Moreover, we conduct the ablation study on the downsampling ratio of the proposed efficient predictor network. Similar to the aforementioned setting, we set our testbed on DnCNN approach and examine three different architectures by setting depth = {3, 6, 15}. We evaluate the runtime speed, FLOPs cost, and PSNR value of four variants with the size of the AvgPooling layer equal to {0, 2, 4, 8}. As shown in Table 2, by processing a 4× downsampled feature map, our proposed efficient predictor network achieves a "win-win" in terms of both performance and efficiency. This demonstrates that applying the prediction network on a lower resolution feature map can not only improve the performance, due to a larger receptive field, but also save computations

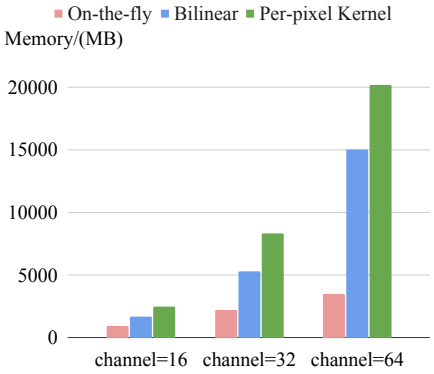


**Fig. 10.** cost comparison between the proposed method and per-pixel kernel prediction approaches (HyperNetwork).

### 4.6 Evaluation on Real Sensor Noise

To further demonstrate the generalization ability of MalleNet, we evaluate our approaches to real sensor noise. Similar to previous works [12,64], we adopt Smartphone Image Denoising Dataset (SIDD) [1] and Darmstadt Noise Dataset (DND) [49] as main benchmarks. We use training data from SIDD as our training set and evaluate our method on both two test sets. In the training process, We adopt Adam Optimizer with a batch size of 128, the weight decay is set to 0.03, and the learning rate is set to 2e-4. We randomly crop 256 × 256 patches and apply random rotation and flipping. We train a real denoiser MalleNet-R, by slightly modifing the channel/depth of MalleNet-M architecture and replace Inverse Bottleneck Block with standard residual block (see supplementary materials for details). As shown in Table 3, MalleNet-R achieves lower latency (**13.58 ms**) compared with other methods. In terms of image quality, MalleNet-R is able to reach similar PSNR/SSIM compared to most baselines, and only slightly behinds the approaches with very heavy computational cost or equipped with complex channel/spatial attention module. We show the PSNR-to-Complexity trade-off of each method in Fig. 12 right. More visual comparisons are included in the supplementary materials.

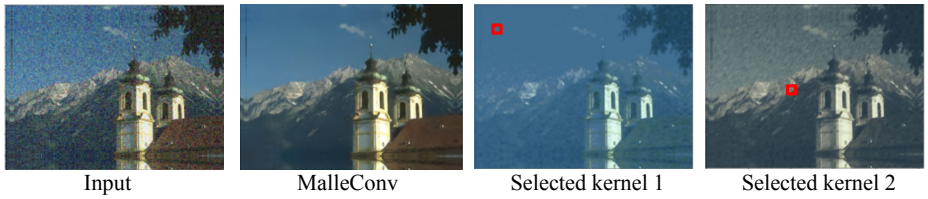


**Fig. 11.** Comparison between default MalleConv output (column2) and outputs using two selected kernels (column 3 and 4).

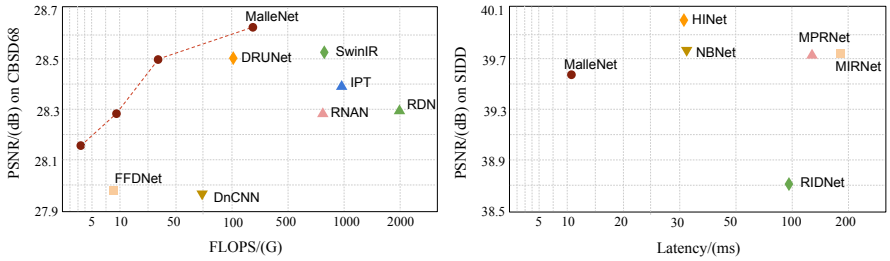


**Fig. 12. Results on CBSD68 test set ($\sigma = 50$) and SIDD validation set.** Our proposed MalleNet architecture achieves a better trade-off between quality and speed.

## 5 Conclusions

In this work, we propose Malleable Convolution (MalleConv), an efficient variant of spatially-varying convolution tailored for ultra-fast image denoising. MalleConv processes a low-resolution feature map and generates a much smaller set of spatially varying filters. The generated filters inherently fit the heterogeneous and spatially varying patterns presented in natural images, while taking little additional computational costs. Despite its effectiveness, we also observe that very deep or wide architectures benefit less from MalleConv, as they may also capture heterogenous image statistics in a less efficient way. Although in this work, we only evaluated MalleConv on image denoising, we believe MalleConv is also capable in other image processing tasks, like dehazing. Another future work is to combine MalleConv with attention mechanism [38] or deformable shape [14] to further improve its quality in applications with less computational constraints.

## 6 Acknowledgement

We would like to express our gratitude to the Google Research Luma team, in particular Zhengzhong Tu for generously providing us with the concrete training recipes on real-world denoising benchmarks.

## References


1. Abdelrahman Abdelhamed, Stephen Lin, and Michael S Brown. A high-quality denoising dataset for smartphone cameras. *CVPR*, 2018.
2. Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. *CVPR workshops*, 2017.
3. Michal Aharon, Michael Elad, and Alfred Bruckstein. K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on signal processing*, 2006.
4. Saeed Anwar and Nick Barnes. Real image denoising with feature attention. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3155–3164, 2019.
5. Steve Bako, Thijs Vogels, Brian McWilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony Derose, and Fabrice Rousselle. Kernel-predicting convolutional networks for denoising monte carlo renderings. *ACM Trans. Graph.*, 2017.
6. Antoni Buades, Bartomeu Coll, and J-M Morel. A non-local algorithm for image denoising. *CVPR*, 2005.
7. Harold C Burger, Christian J Schuler, and Stefan Harmeling. Image denoising: Can plain neural networks compete with bm3d? *CVPR*, 2012.
8. Hanting Chen, Yunhe Wang, Tianyu Guo, Chang Xu, Yiping Deng, Zhenhua Liu, Siwei Ma, Chunjing Xu, Chao Xu, and Wen Gao. Pre-trained image processing transformer. *CVPR*, 2021.
9. Liangyu Chen, Xin Lu, Jie Zhang, Xiaojie Chu, and Chengpeng Chen. Hinet: Half instance normalization network for image restoration. *CVPR*, 2021.
10. Yunjin Chen and Thomas Pock. Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration. *TPAMI*, 2016.
11. Zeyuan Chen, Yifan Jiang, Dong Liu, and Zhangyang Wang. Cerl: A unified optimization framework for light enhancement with realistic noise. *IEEE Transactions on Image Processing*, 2022.
12. Shen Cheng, Yuzhi Wang, Haibin Huang, Donghao Liu, Haoqiang Fan, and Shuaicheng Liu. Nbnet: Noise basis learning for image denoising with subspace projection. *CVPR*, 2021.
13. Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. Image denoising by sparse 3-d transform-domain collaborative filtering. *IEEE Transactions on image processing*, 2007.
14. Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 764–773, 2017.
15. Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *TPAMI*, 2015.
16. Michael Elad and Michal Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Transactions on Image processing*, 2006.
17. Rich Franzen. Kodak lossless true color image suite. *source: http://r0k.us/graphics/kodak*, 1999.
18. Pascal Getreuer, Ignacio Garcia-Dorado, John Isidoro, Sungjoon Choi, Frank Ong, and Peyman Milanfar. Blade: Filter learning for general purpose computational photography. In *2018 IEEE International Conference on Computational Photography (ICCP)*, pages 1–11. IEEE, 2018.
19. Michaël Gharbi, Jiawen Chen, Jonathan T. Barron, Samuel W Hasinoff, and Frédo Durand. Deep bilateral learning for real-time image enhancement. *SIGGRAPH*, 2017.

20. Daniel Glasner, Shai Bagon, and Michal Irani. Super-resolution from a single image. *ICCV*, 2009.
21. Shuhang Gu, Wen Li, Luc Van Gool, and Radu Timofte. Fast image restoration with multi-bin trainable linear units. *ICCV*, 2019.
22. Shuhang Gu, Yawei Li, Luc Van Gool, and Radu Timofte. Self-guided network for fast image denoising. *ICCV*, 2019.
23. Shuhang Gu, Lei Zhang, Wangmeng Zuo, and Xiangchu Feng. Weighted nuclear norm minimization with application to image denoising. *CVPR*, 2014.
24. Shi Guo, Zifei Yan, Kai Zhang, Wangmeng Zuo, and Lei Zhang. Toward convolutional blind denoising of real photographs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1712–1722, 2019.
25. David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv:1609.09106*, 2016.
26. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *ICCV*, 2015.
27. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, 2016.
28. Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017.
29. Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. *NeurIPS*, 2016.
30. Yifan Jiang, Xinyu Gong, Ding Liu, Yu Cheng, Chen Fang, Xiaohui Shen, Jianchao Yang, Pan Zhou, and Zhangyang Wang. Enlightengan: Deep light enhancement without paired supervision. *IEEE Transactions on Image Processing*, 2021.
31. Yifan Jiang, He Zhang, Jianming Zhang, Yilin Wang, Zhe Lin, Kalyan Sunkavalli, Simon Chen, Sohrab Amirghodsi, Sarah Kong, and Zhangyang Wang. Ssh: A self-supervised framework for image harmonization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4832–4841, 2021.
32. Hirokatsu Kataoka, Kazushige Okayasu, Asato Matsumoto, Eisuke Yamagata, Ryosuke Yamada, Nakamasa Inoue, Akio Nakamura, and Yutaka Satoh. Pre-training without natural images. *ACCV*, 2020.
33. Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.
34. Orest Kupyn, Volodymyr Budzan, Mykola Mykhailych, Dmytro Mishkin, and Jiří Matas. Deblurgan: Blind motion deblurring using conditional adversarial networks. *CVPR*, 2018.
35. Orest Kupyn, Tetiana Martyniuk, Junru Wu, and Zhangyang Wang. Deblurgan-v2: Deblurring (orders-of-magnitude) faster and better. *ICCV*, 2019.
36. Duo Li, Jie Hu, Changhu Wang, Xiangtai Li, Qi She, Lei Zhu, Tong Zhang, and Qifeng Chen. Involution: Inverting the inherence of convolution for visual recognition. *CVPR*, 2021.
37. Siyuan Li, Iago Breno Araujo, Wenqi Ren, Zhangyang Wang, Eric K Tokuda, Roberto Hirata Junior, Roberto Cesar-Junior, Jiawan Zhang, Xiaojie Guo, and Xiaochun Cao. Single image deraining: A comprehensive benchmark analysis. *CVPR*, 2019.
38. Jingyun Liang, Jiezhang Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. Swinir: Image restoration using swin transformer. *ICCV*, 2021.
39. Jie Liang, Hui Zeng, and Lei Zhang. High-resolution photorealistic image translation in real-time: A laplacian pyramid translation network. *CVPR*, 2021.
40. Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. *CVPR workshops*, 2017.

41. Xudong Lin, Lin Ma, Wei Liu, and Shih-Fu Chang. Context-gated convolution. *ECCV*, 2020.
42. Kede Ma, Zhengfang Duanmu, Qingbo Wu, Zhou Wang, Hongwei Yong, Hongliang Li, and Lei Zhang. Waterloo exploration database: New challenges for image quality assessment models. *IEEE Transactions on Image Processing*, 2016.
43. Julien Mairal, Francis Bach, Jean Ponce, Guillermo Sapiro, and Andrew Zisserman. Non-local sparse models for image restoration. *ICCV*, 2009.
44. D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001.
45. Ben Mildenhall, Jonathan T. Barron, Jiawen Chen, Dillon Sharlet, Ren Ng, and Robert Carroll. Burst denoising with kernel prediction networks. *CVPR*, 2018.
46. Yali Peng, Lu Zhang, Shigang Liu, Xiaojun Wu, Yu Zhang, and Xili Wang. Dilated residual networks with symmetric skip connection for image denoising. *Neurocomputing*, 2019.
47. Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *TPAMI*, 1990.
48. Gabriel Peyré, Sébastien Bougleux, and Laurent Cohen. Non-local regularization of inverse problems. *ECCV*, 2008.
49. Tobias Plotz and Stefan Roth. Benchmarking denoising algorithms with real photographs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1586–1595, 2017.
50. Dongwei Ren, Wangmeng Zuo, Qinghua Hu, Pengfei Zhu, and Deyu Meng. Progressive image deraining networks: A better and simpler baseline. *CVPR*, 2019.
51. Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena*, 1992.
52. Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *CVPR*, 2018.
53. Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1874–1883, 2016.
54. Xin Tao, Hongyun Gao, Xiaoyong Shen, Jue Wang, and Jiaya Jia. Scale-recurrent network for deep image deblurring. *CVPR*, 2018.
55. Chunwei Tian, Yong Xu, and Wangmeng Zuo. Image denoising using deep cnn with batch renormalization. *Neural Networks*, 2020.
56. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 2017.
57. Jiaqi Wang, Kai Chen, Rui Xu, Ziwei Liu, Chen Change Loy, and Dahua Lin. Carafe: Content-aware reassembly of features. *ICCV*, 2019.
58. Ze Wang, Zichen Miao, Jun Hu, and Qiang Qiu. Adaptive convolutions with per-pixel dynamic filter atom. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12302–12311, 2021.
59. Chen Wei, Wenjing Wang, Wenhan Yang, and Jiaying Liu. Deep retinex decomposition for low-light enhancement. *arXiv:1808.04560*, 2018.
60. Zhihao Xia and Ayan Chakrabarti. Identifying recurring patterns with deep neural networks for natural image denoising. *WACV*, 2020.
61. Yu-Syuan Xu, Shou-Yao Roy Tseng, Yu Tseng, Hsien-Kai Kuo, and Yi-Min Tsai. Unified dynamic convolutional network for super-resolution with variational degradations. *CVPR*, 2020.

62. Wenhan Yang, Robby T Tan, Shiqi Wang, Yuming Fang, and Jiaying Liu. Single image deraining: From model-based to data-driven and beyond. *TPAMI*, 2020.
63. Zongsheng Yue, Hongwei Yong, Qian Zhao, Lei Zhang, and Deyu Meng. Variational denoising network: Toward blind noise modeling and removal. *arXiv preprint arXiv:1908.11314*, 2019.
64. Syed Waqas Zamir, Aditya Arora, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, Ming-Hsuan Yang, and Ling Shao. Learning enriched features for real image restoration and enhancement. *ECCV*, 2020.
65. Syed Waqas Zamir, Aditya Arora, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, Ming-Hsuan Yang, and Ling Shao. Multi-stage progressive image restoration. *CVPR*, 2021.
66. Kai Zhang, Yawei Li, Wangmeng Zuo, Lei Zhang, Luc Van Gool, and Radu Timofte. Plug-and-play image restoration with deep denoiser prior. *TPAMI*, 2021.
67. Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE transactions on image processing*, 2017.
68. Kai Zhang, Wangmeng Zuo, Shuhang Gu, and Lei Zhang. Learning deep cnn denoiser prior for image restoration. *CVPR*, 2017.
69. Kai Zhang, Wangmeng Zuo, and Lei Zhang. Ffdnet: Toward a fast and flexible solution for cnn-based image denoising. *IEEE Transactions on Image Processing*, 2018.
70. Lei Zhang, Xiaolin Wu, Antoni Buades, and Xin Li. Color demosaicking by local directional interpolation and nonlocal adaptive thresholding. *Journal of Electronic imaging*, 2011.
71. Yulun Zhang, Kunpeng Li, Kai Li, Bineng Zhong, and Yun Fu. Residual non-local attention networks for image restoration. *arXiv:1903.10082*, 2019.
72. Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. Residual dense network for image super-resolution. *CVPR*, 2018.
73. Yulun Zhang, Donglai Wei, Can Qin, Huan Wang, Hanspeter Pfister, and Yun Fu. Context reasoning attention network for image super-resolution. *ICCV*, 2021.