RESTORMER: EFFICIENT TRANSFORMER FOR HIGH-RESOLUTION IMAGE RESTORATION

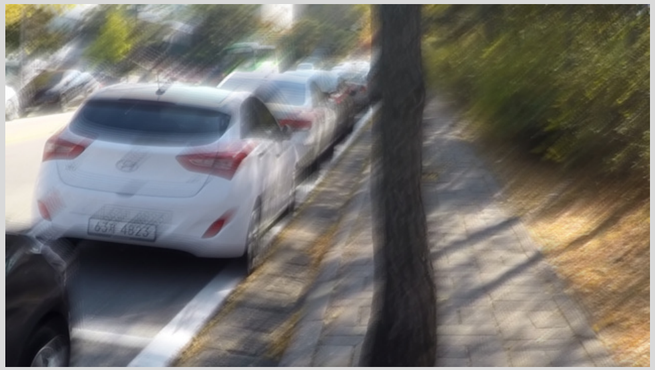
by

Syed Waqas Zamir, Aditya Arora, Salman Khan, Munawar Hayat
Fahad Shahbaz Khan, Ming-Hsuan Yang

THE PROBLEM

Image Restoration

Operation of taking a corrupt/noisy image and estimating the clean, original image.

Image Deraining	Motion Deblurring	Defocus Deblurring	Image Denoising
			
			

THE REAL PROBLEM

Image Restoration

- Transformers, have shown signifi- cant performance gains on natural language and high-level vision tasks (ViT).

- Computational complexity grows quadratically with the spatial resolution. Infeasible for most image restoration tasks with higher resolution.

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

- How to reduce the computational complexity of trasformers while maintaing the performance.

ARCHITECTURE

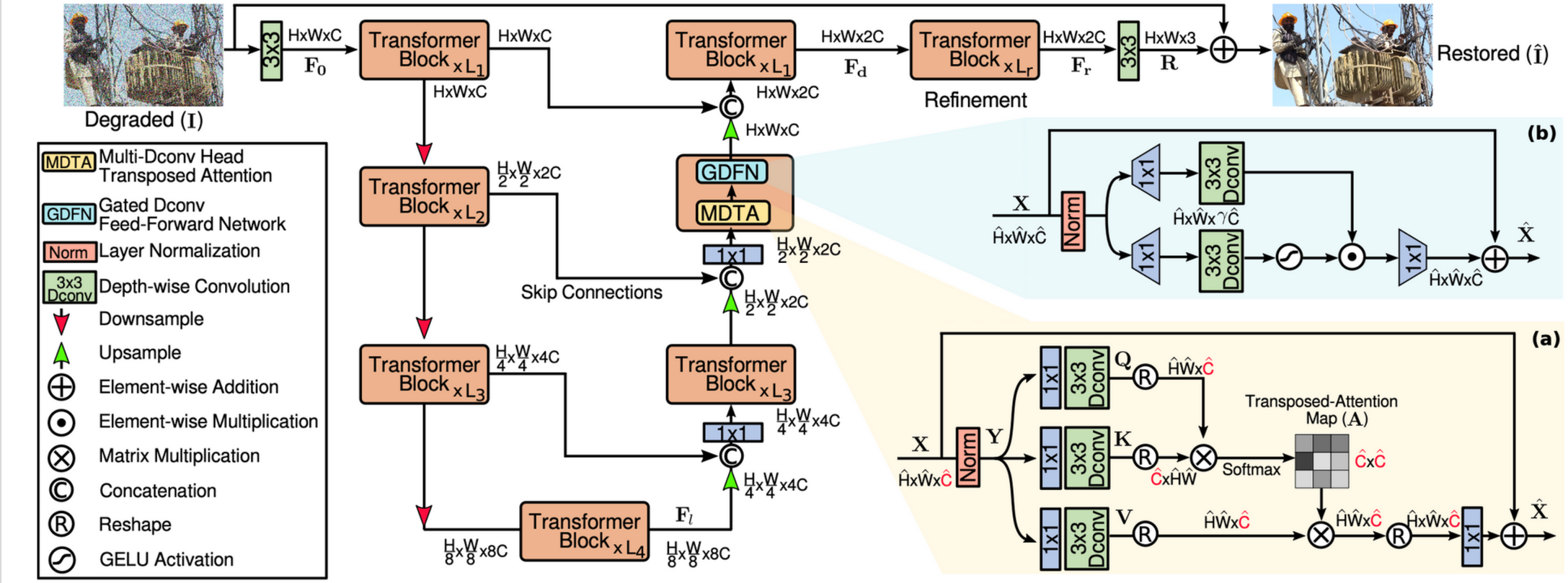
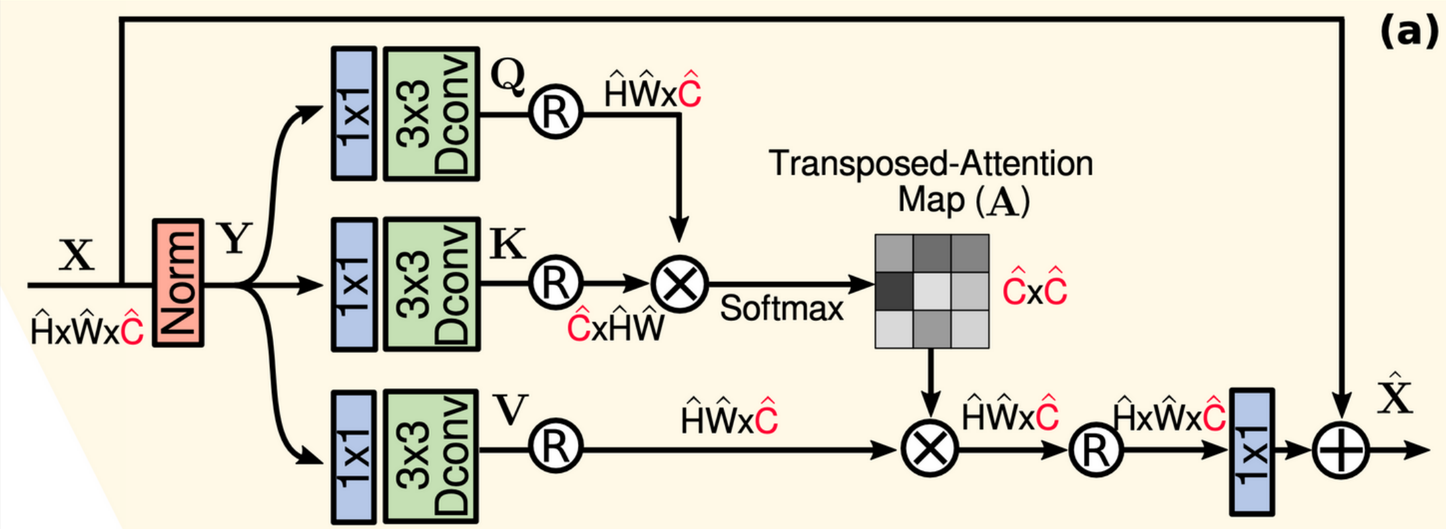


Figure 2. Architecture of Restormer for high-resolution image restoration. Our Restormer consists of multiscale hierarchical design incorporating efficient Transformer blocks. The core modules of Transformer block are: **(a)** multi-Dconv head transposed attention (MDTA) that performs (spatially enriched) *query-key* feature interaction across channels rather the spatial dimension, and **(b)** Gated-Dconv feed-forward network (GDFN) that performs controlled feature transformation, *i.e.*, to allow useful information to propagate further.

Goal - "... propose an efficient Transformer for image restoration that is capable of modeling global connectivity and is still applicable to large images."

ARCHITECTURE

1) Multi Dconv Head Transpose Attention (MDTA)



$$\hat{\mathbf{X}} = W_p\,\text{Attention}\left(\hat{\mathbf{Q}}, \hat{\mathbf{K}}, \hat{\mathbf{V}}\right) + \mathbf{X},$$

$$\text{Attention}\left(\hat{\mathbf{Q}}, \hat{\mathbf{K}}, \hat{\mathbf{V}}\right) = \hat{\mathbf{V}} \cdot \text{Softmax}\left(\hat{\mathbf{K}} \cdot \hat{\mathbf{Q}}/\alpha\right),$$

1. Normalized tensor $\mathbf{Y} \in \mathbb{R}^{\hat{H}\times\hat{W}\times\hat{C}}$
2. 1x1 convolution and 3x3 convolution for local context mixing. and Generating Q,K,V projections. $\mathbf{Q}=W_d^Q W_p^Q \mathbf{Y}$, $\mathbf{K}=W_d^K W_p^K \mathbf{Y}$ and $\mathbf{V}=W_d^V W_p^V \mathbf{Y}$.
3. Reshaping the tensors from HxWxC such that the dot product from Q and K generates a transposed attention map A of size CxC. $\hat{\mathbf{Q}} \in \mathbb{R}^{\hat{H}\hat{W}\times\hat{C}}$; $\hat{\mathbf{K}} \in \mathbb{R}^{\hat{C}\times\hat{H}\hat{W}}$; and $\hat{\mathbf{V}} \in \mathbb{R}^{\hat{H}\hat{W}\times\hat{C}}$
4. Alpha - Learnable scaling parameter.
5. Skip connections to learn the residue.

ARCHITECTURE

1) Multi Dconv Head Transpose Attention (MDTA)

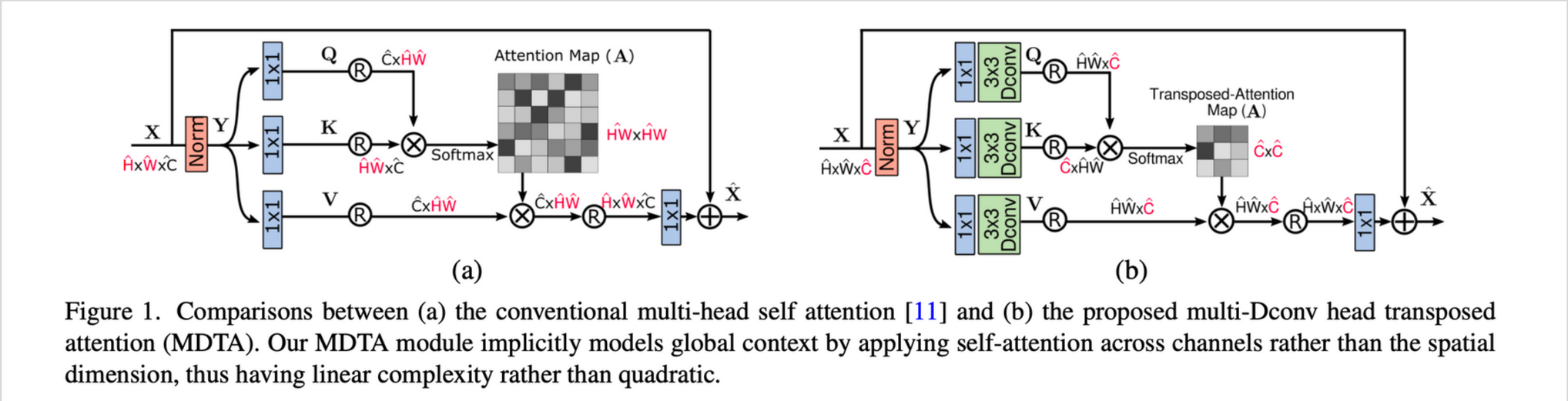


Figure 1. Comparisons between (a) the conventional multi-head self attention [11] and (b) the proposed multi-Dconv head transposed attention (MDTA). Our MDTA module implicitly models global context by applying self-attention across channels rather than the spatial dimension, thus having linear complexity rather than quadratic.

What's New In MDTA

- Local contest mixing before feature covariance computation.
  - **1x1 convolution** - Pixel-wise aggregation of cross channel context
  - **3x3 convolution** - Channel-wise aggregation of local context
- Applies SA across feature dimension rather than the spatial dimension.
- Computes cross-covariance across features channels instead of modeling pairwise pixel interactions.

ARCHITECTURE

1) Multi Dconv Head Transpose Attention (MDTA)

Reasons for MDTA to Work Well

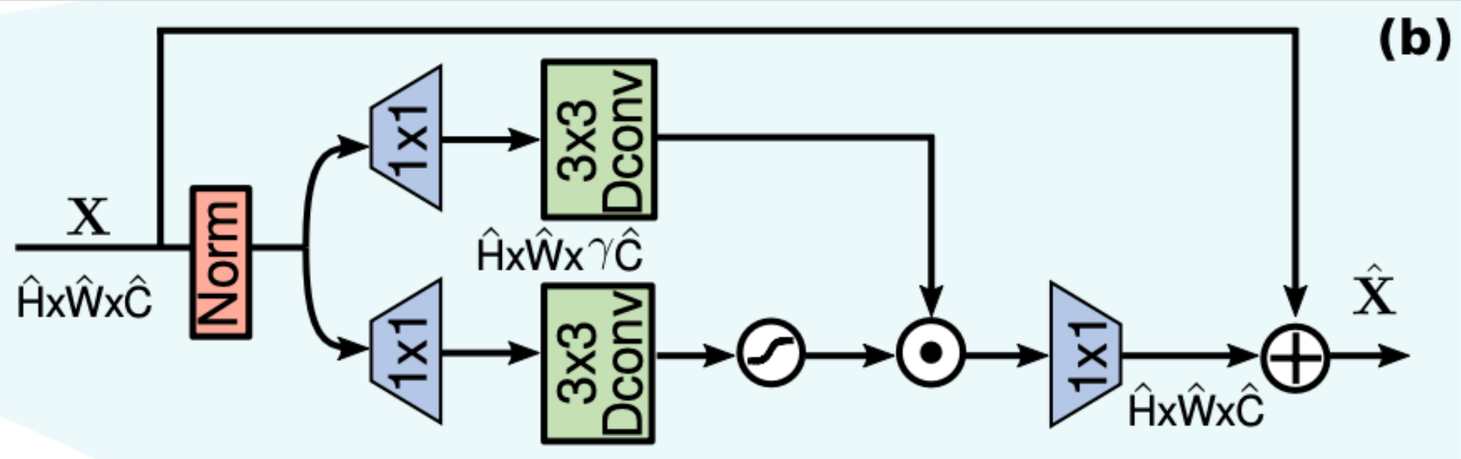
- Emphazisation on the spacially local context which brings in the strength in convolution operation and its inductive bias nature within the pipeline.

- Contextualised global relationships between pixels are implicitly modeled when computing covariance based attention.

```python
class MDTA(nn.Module):
    def __init__(self, channels, num_heads):
        super(MDTA, self).__init__()
        self.num_heads = num_heads
        self.temperature = nn.Parameter(torch.ones(1, num_heads, 1, 1))

        self.qkv = nn.Conv2d(channels, channels * 3, kernel_size=1, bias=False)
        self.qkv_conv = nn.Conv2d(channels * 3, channels * 3, kernel_size=3, padding=1, groups=channels * 3, bias=False)
        self.project_out = nn.Conv2d(channels, channels, kernel_size=1, bias=False)

    def forward(self, x):
        b, c, h, w = x.shape
        q, k, v = self.qkv_conv(self.qkv(x)).chunk(3, dim=1)

        q = q.reshape(b, self.num_heads, -1, h * w)
        k = k.reshape(b, self.num_heads, -1, h * w)
        v = v.reshape(b, self.num_heads, -1, h * w)
        q, k = F.normalize(q, dim=-1), F.normalize(k, dim=-1)

        attn = torch.softmax(torch.matmul(q, k.transpose(-2, -1).contiguous()) * self.temperature, dim=-1)
        out = self.project_out(torch.matmul(attn, v).reshape(b, -1, h, w))
        return out
```

ARCHITECTURE

2) Gated D-conv Feed Forward Network (GDFN)



$$\hat{\mathbf{X}} = W_p^0 \,\text{Gating}\,(\mathbf{X}) + \mathbf{X},$$
$$\text{Gating}(\mathbf{X}) = \phi(W_d^1 W_p^1(\text{LN}(\mathbf{X}))) \odot W_d^2 W_p^2(\text{LN}(\mathbf{X}))$$

1. Input Tensor $\mathbf{X} \in \mathbb{R}^{\hat{H} \times \hat{W} \times \hat{C}}$
2. **Depth wise Convolutions** - 1x1 convolution and 3x3 convolution for local context mixing to encode information from spatially neighbouring pixel positions.$W_d^1 W_p^1(\text{LN}(\mathbf{X}))$
3. **Gating Mechanism** - Element-wise product of two parallel paths of linear transformation layers, one of which is activated with the GELU non-linearity $\text{Gating}(\mathbf{X}) = \phi(W_d^1 W_p^1(\text{LN}(\mathbf{X}))) \odot W_d^2 W_p^2(\text{LN}(\mathbf{X})$
4. 1x1 Convolution - The first 1x1 expands the feature channels and the second 1x1 reduces the channel count to original
5. A skip connection.

ARCHITECTURE

2) Gated D-conv Feed Forward Network (GDFN)

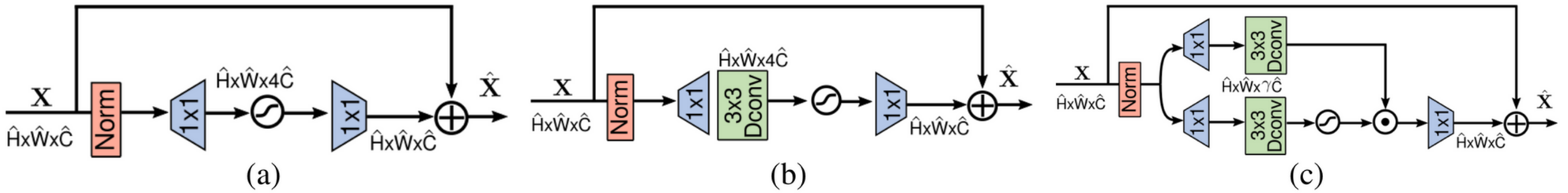


Figure 2. Comparisons among (a) conventional feed-forward network [11], (b) Dconv feed-forward network [21] and (c) the proposed gated-Dconv feed-forward network (GDFN). Since the proposed GDFN performs more operations as compared to (a) and (b), we reduce the expansion ratio γ so as to have similar parameters and identical compute burden.

What's New In GDTA

- **Element wise product** of two linear layers, one of which is activated with GELU non- linearity. Which in tern act as a **gating mechanism.**

ARCHITECTURE

2) Gated D-conv Feed Forward Network (GDFN)

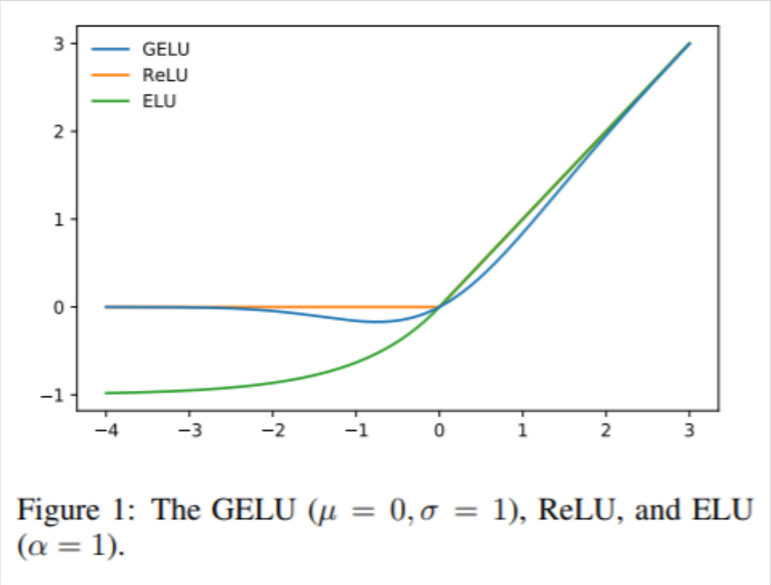


Figure 1: The GELU ($\mu = 0, \sigma = 1$), ReLU, and ELU ($\alpha = 1$).

Why GeLU Activation.

- ReLU can suffer from "problems where significant amount of neuron in the network become zero."
- GelU is **smoother near zero** and is **differentiable in all ranges**, and allows to have gradients (although small) in negative range.

ARCHITECTURE

2) Gated D-conv Feed Forward Network (GDFN)

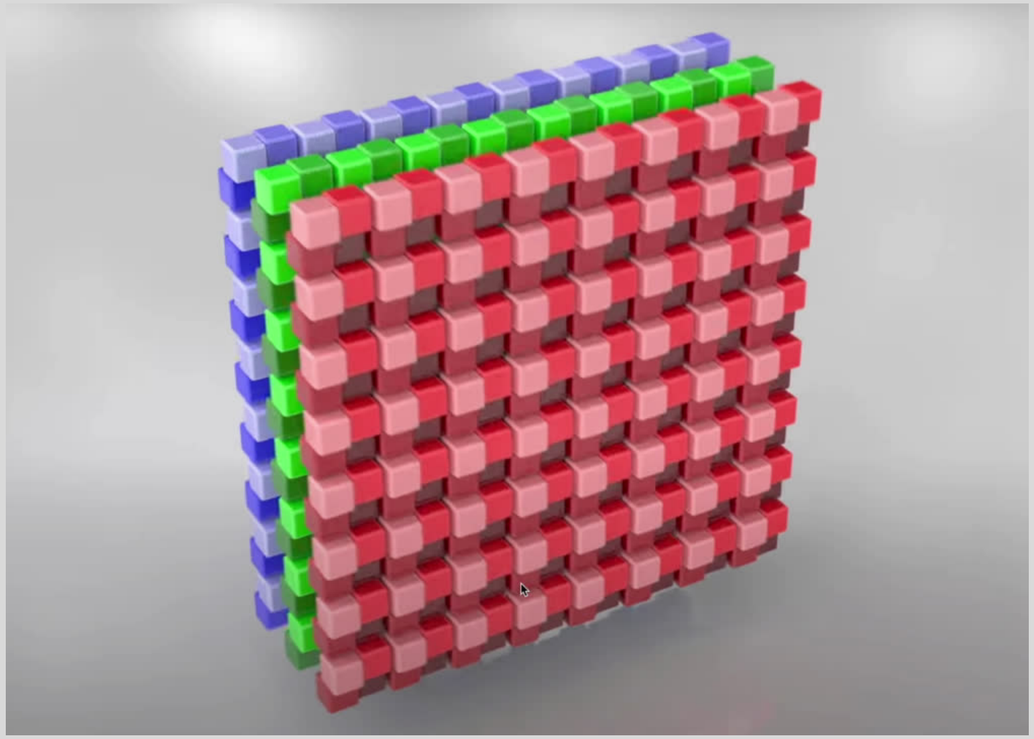
Reasons for GDFN to Work Well

- Controlling which complementary features should flow forward through the respective hierarchical level.
- Allowing subsequent layers to specifically focus on more refined attributes to that specific level.

```
class GDFN(nn.Module):
    def __init__(self, channels, expansion_factor):
        super(GDFN, self).__init__()

        hidden_channels = int(channels * expansion_factor)
        self.project_in = nn.Conv2d(channels, hidden_channels * 2, kernel_size=1, bias=False)
        self.conv = nn.Conv2d(hidden_channels * 2, hidden_channels * 2, kernel_size=3, padding=1,
                              groups=hidden_channels * 2, bias=False)
        self.project_out = nn.Conv2d(hidden_channels, channels, kernel_size=1, bias=False)

    def forward(self, x):
        x1, x2 = self.conv(self.project_in(x)).chunk(2, dim=1)
        x = self.project_out(F.gelu(x1) * x2)
        return x
```

ARCHITECTURE

3) Pixel Shuffle and Pixel Unshuffle.



PixelShuffle is an operation used in super-resolution models to implement efficient sub-pixel convolutions with a stride of $1/r$. Specifically it rearranges elements in a tensor of shape $(*, C \times r^2, H, W)$ to a tensor of shape $(*, C, H \times r, W \times r)$.

ARCHITECTURE

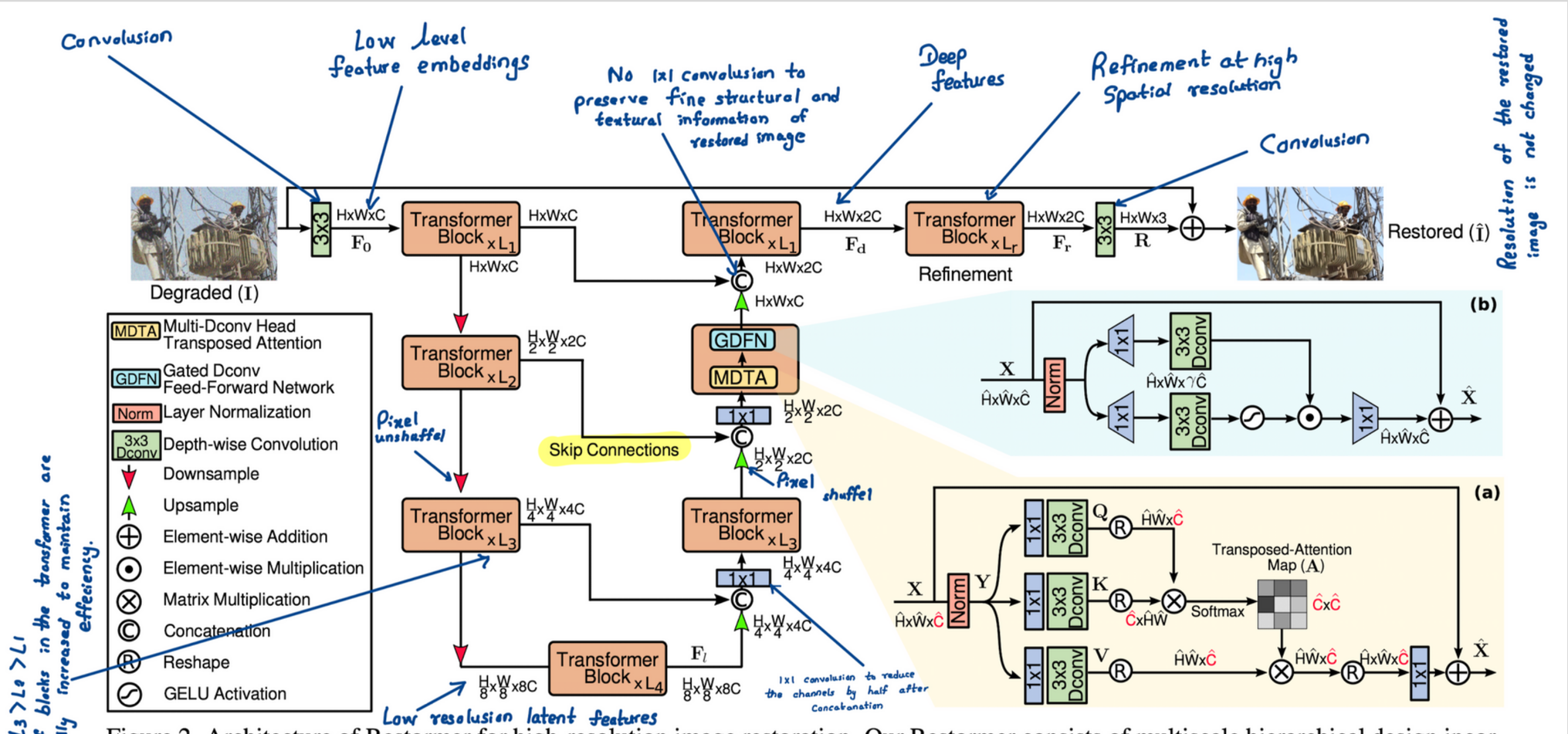


Figure 2. Architecture of Restormer for high-resolution image restoration. Our Restormer consists of multiscale hierarchical design incorporating efficient Transformer blocks. The core modules of Transformer block are: **(a)** multi-Dconv head transposed attention (MDTA) that performs (spatially enriched) *query-key* feature interaction across channels rather the spatial dimension, and **(b)** Gated-Dconv feed-forward network (GDFN) that performs controlled feature transformation, *i.e.*, to allow useful information to propagate further.

ARCHITECTURE

- Degraded image. $\mathbf{I} \in \mathbb{R}^{H \times W \times 3}$
- **Convolution** to obtain low-level feature embeddings.$\mathbf{F_0} \in \mathbb{R}^{H \times W \times C}$
- Hierarchically reduce spatial size while expanding channel capacity using **pixel-shuffle**.
- Decoder takes low resolution latent features. $\mathbf{F}_l \in \mathbb{R}^{\frac{H}{8} \times \frac{W}{8} \times 8C}$
- Converting low level features into deap features using 4 level **symmetric enocder-decoder** transformer.$\mathbf{F_d} \in \mathbb{R}^{H \times W \times 2C}$
- **Skip connections** - skip connections are concatenated to assist the recovery process.
- **1x1 convlution** - Reduce channel count by half in all levels except the first one.
- **At level-1 Transformer blocks aggregate the low-level image features of the encoder with the high-level features of the decoder** - Benificial in preserving the fine structural and textural details in the restored image.
- **Refinement Stage** - To refine at high spatial resolution.
- **Convolution layer** - To generate residual image.

IMPLEMENTATION DETAILS

Datasets Used

Tasks	Datasets	Train Samples	Test Samples	Testset Rename
Deraining	Rain14000 [13]	11200	2800	Test2800
	Rain1800 [36]	1800	0	-
	Rain800 [42]	700	100	Test100
	Rain100H [36]	0	100	Rain100H
	Rain100L [36]	0	100	Rain100L
	Rain1200 [41]	0	1200	Test1200
	Rain12 [20]	12	0	-
Motion Deblurring	GoPro [25]	2103	1111	-
	HIDE [32]	0	2025	-
	RealBlur [31]	0	1960	-
Denoising	SIDD [1]	320	1280 patches from 40 images	-
	DND [27]	0	1000 patches from 50 images	-
	DIV2K [4]	800	0	-
	Flickr2K	2650	0	-
	BSD500 [5]	400	0	-
	WED [23]	4744	0	-
	Set12 [45]	0	12	-
	BSD68 [24]	0	68	-
	Urban100 [15]	0	100	-
	Kodak24 [12]	0	24	-
	McMaster [46]	0	18	-
Defocus Deblurring	DPDD [2]	350	76	-

IMPLEMENTATION DETAILS

Progressive Learning

- Network is trained on smaller image patches in the early epochs and on grad- ually larger patches in the later training epochs
- Large patches comes at the cost of longer time, the batch size is reduced as the patch size increases to maintain a similar time per optimization.
- **Reason for this to work -** Training a Transformer model on small cropped patches may not encode the global image statistics

EVALUATION MATRICES

PSNR

```python
def psnr(x, y, data_range=255.0):
    x, y = x / data_range, y / data_range
    mse = torch.mean((x - y) ** 2)
    score = - 10 * torch.log10(mse)
    return score
```

$$PSNR = 10log10\left(\frac{max^2}{MSE}\right)$$

EVALUATION MATRICES

SSIM

```python
def ssim(x, y, kernel_size=11, kernel_sigma=1.5, data_range=255.0, k1=0.01, k2=0.03):
    x, y = x / data_range, y / data_range
    # average pool image if the size is large enough
    f = max(1, round(min(x.size()[-2:]) / 256))
    if f > 1:
        x, y = F.avg_pool2d(x, kernel_size=f), F.avg_pool2d(y, kernel_size=f)

    # gaussian filter
    coords = torch.arange(kernel_size, dtype=x.dtype, device=x.device)
    coords -= (kernel_size - 1) / 2.0
    g = coords ** 2
    g = (- (g.unsqueeze(0) + g.unsqueeze(1)) / (2 * kernel_sigma ** 2)).exp()
    g /= g.sum()
    kernel = g.unsqueeze(0).repeat(x.size(1), 1, 1, 1)

    # compute
    c1, c2 = k1 ** 2, k2 ** 2
    n_channels = x.size(1)
    mu_x = F.conv2d(x, weight=kernel, stride=1, padding=0, groups=n_channels)
    mu_y = F.conv2d(y, weight=kernel, stride=1, padding=0, groups=n_channels)

    mu_xx, mu_yy, mu_xy = mu_x ** 2, mu_y ** 2, mu_x * mu_y
    sigma_xx = F.conv2d(x ** 2, weight=kernel, stride=1, padding=0, groups=n_channels) - mu_xx
    sigma_yy = F.conv2d(y ** 2, weight=kernel, stride=1, padding=0, groups=n_channels) - mu_yy
    sigma_xy = F.conv2d(x * y, weight=kernel, stride=1, padding=0, groups=n_channels) - mu_xy

    # contrast sensitivity (CS) with alpha = beta = gamma = 1.
    cs = (2.0 * sigma_xy + c2) / (sigma_xx + sigma_yy + c2)
    # structural similarity (SSIM)
    ss = (2.0 * mu_xy + c1) / (mu_xx + mu_yy + c1) * cs
    return ss.mean()
```

Non-linear full-reference metric that compares the luminance, contrast and structure of the original and degraded image

[PSNR vs SSIM: imperceptibility quality assessment
De Rosal Igantius Moses Setiadi]

EXPERIMENT AND ANALYSIS

Image Deraining

Table 1. **Image deraining** results. When averaged across all five datasets, our Restormer advances state-of-the-art by 1.05 dB.

Method	Test100 [97]		Rain100H [86]		Rain100L [86]		Test2800 [22]		Test1200 [96]		Average	
	PSNR ↑	SSIM ↑	PSNR ↑	SSIM ↑	PSNR ↑	SSIM ↑	PSNR ↑	SSIM ↑	PSNR ↑	SSIM ↑	PSNR ↑	SSIM ↑
DerainNet [21]	22.77	0.810	14.92	0.592	27.03	0.884	24.31	0.861	23.38	0.835	22.48	0.796
SEMI [81]	22.35	0.788	16.56	0.486	25.03	0.842	24.43	0.782	26.05	0.822	22.88	0.744
DIDMDN [96]	22.56	0.818	17.35	0.524	25.23	0.741	28.13	0.867	29.65	0.901	24.58	0.770
UMRL [87]	24.41	0.829	26.01	0.832	29.18	0.923	29.97	0.905	30.55	0.910	28.02	0.880
RESCAN [43]	25.00	0.835	26.36	0.786	29.80	0.881	31.29	0.904	30.51	0.882	28.59	0.857
PreNet [64]	24.81	0.851	26.77	0.858	32.44	0.950	31.75	0.916	31.36	0.911	29.42	0.897
MSPFN [32]	27.50	0.876	28.66	0.860	32.40	0.933	32.82	0.930	32.39	0.916	30.75	0.903
MPRNet [93]	30.27	0.897	30.41	0.890	36.40	0.965	33.64	0.938	32.91	0.916	32.73	0.921
SPAIR [61]	30.35	0.909	30.95	0.892	36.93	0.969	33.34	0.936	33.04	0.922	32.91	0.926
Restormer	**32.00**	**0.923**	**31.46**	**0.904**	**38.99**	**0.978**	**34.18**	**0.944**	**33.19**	**0.926**	**33.96**	**0.935**

- Compared to the recent best method SPAIR, Restormer achieves 1.05 dB improvement when averaged across all datasets.

EXPERIMENT AND ANALYSIS

Image Deraining

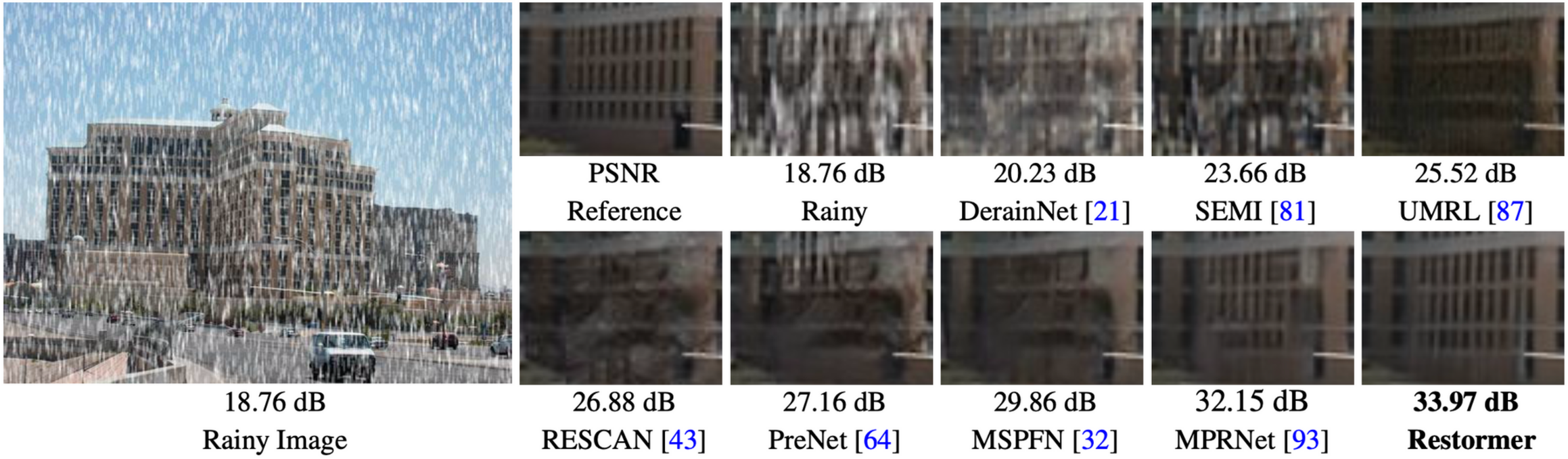


Figure 3. **Image deraining** example. Our Restormer generates rain-free image with structural fidelity and without artifacts.

EXPERIMENT AND ANALYSIS

Motion Deblurring

Table 2. **Single-image motion deblurring** results. Our Restormer is trained only on the GoPro dataset [56] and directly applied to the HIDE [67] and RealBlur [65] benchmark datasets.

Method	GoPro [56]		HIDE [67]		RealBlur-R [65]		RealBlur-J [65]	
	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
Xu *et al.* [84]	21.00	0.741	-		34.46	0.937	27.14	0.830
DeblurGAN [38]	28.70	0.858	24.51	0.871	33.79	0.903	27.97	0.834
Nah *et al.* [56]	29.08	0.914	25.73	0.874	32.51	0.841	27.87	0.827
Zhang *et al.* [98]	29.19	0.931	-		35.48	0.947	27.80	0.847
DeblurGAN-v2 [39]	29.55	0.934	26.61	0.875	35.26	0.944	28.70	0.866
SRN [72]	30.26	0.934	28.36	0.915	35.66	0.947	28.56	0.867
Shen *et al.* [67]	-		28.89	0.930	-		-	
Gao *et al.* [23]	30.90	0.935	29.11	0.913	-		-	
DBGAN [100]	31.10	0.942	28.94	0.915	33.78	0.909	24.93	0.745
MT-RNN [58]	31.15	0.945	29.15	0.918	35.79	0.951	28.44	0.862
DMPHN [94]	31.20	0.940	29.09	0.924	35.70	0.948	28.42	0.860
Suin *et al.* [71]	31.85	0.948	29.98	0.930	-		-	
SPAIR [61]	32.06	0.953	30.29	0.931	-		28.81	0.875
MIMO-UNet+ [14]	32.45	0.957	29.99	0.930	35.54	0.947	27.63	0.837
IPT [13]	32.52	-	-		-		-	
MPRNet [93]	32.66	0.959	30.96	0.939	35.99	0.952	28.70	0.873
Restormer	**32.92**	**0.961**	**31.22**	**0.942**	**36.19**	**0.957**	**28.96**	**0.879**

- Performance boost of 0.47 dB over the recent algorithm MIMO-UNet+
- Performance boost of0.26 dB over the previous best method MPRNet
- Restomer is trained only on the train set of GoPro dataset. Hence shows **strong generalization.**

EXPERIMENT AND ANALYSIS

Motion Deblurring

19.45 dB
Blurry Image

PSNR
Reference

19.45 dB
Blurry

23.85 dB
Gao *et al*. [23]

23.56 dB
DBGAN [100]

23.86 dB
MTRNN [58]

24.85 dB
DMPHN [94]

25.20 dB
Suin *et al*. [71]

25.67 dB
MPRNet [93]

24.33 dB
MIMO-UNet+ [14]

26.96 dB
Restormer

Figure 4. **Single image motion deblurring** on GoPro [56]. Restormer generates sharper and visually-faithful result.

EXPERIMENT AND ANALYSIS

Defocus Deblurring

Table 3. **Defocus deblurring** comparisons on the DPDD testset [3] (containing 37 indoor and 39 outdoor scenes). **S:** single-image defocus deblurring. **D:** dual-pixel defocus deblurring. Restormer sets new state-of-the-art for both single-image and dual pixel defocus deblurring.

Method	Indoor Scenes				Outdoor Scenes				Combined			
	PSNR ↑	SSIM ↑	MAE ↓	LPIPS ↓	PSNR ↑	SSIM ↑	MAE ↓	LPIPS ↓	PSNR ↑	SSIM ↑	MAE ↓	LPIPS ↓
EBDB$_S$ [33]	25.77	0.772	0.040	0.297	21.25	0.599	0.058	0.373	23.45	0.683	0.049	0.336
DMENet$_S$ [40]	25.50	0.788	0.038	0.298	21.43	0.644	0.063	0.397	23.41	0.714	0.051	0.349
JNB$_S$ [68]	26.73	0.828	0.031	0.273	21.10	0.608	0.064	0.355	23.84	0.715	0.048	0.315
DPDNet$_S$ [3]	26.54	0.816	0.031	0.239	22.25	0.682	0.056	0.313	24.34	0.747	0.044	0.277
KPAC$_S$ [70]	27.97	0.852	0.026	0.182	22.62	0.701	0.053	0.269	25.22	0.774	0.040	0.227
IFAN$_S$ [41]	28.11	0.861	0.026	0.179	22.76	0.720	0.052	0.254	25.37	0.789	0.039	0.217
Restormer$_S$	**28.87**	**0.882**	**0.025**	**0.145**	**23.24**	**0.743**	**0.050**	**0.209**	**25.98**	**0.811**	**0.038**	**0.178**
DPDNet$_D$ [3]	27.48	0.849	0.029	0.189	22.90	0.726	0.052	0.255	25.13	0.786	0.041	0.223
RDPD$_D$ [4]	28.10	0.843	0.027	0.210	22.82	0.704	0.053	0.298	25.39	0.772	0.040	0.255
Uformer$_D$ [80]	28.23	0.860	0.026	0.199	23.10	0.728	0.051	0.285	25.65	0.795	0.039	0.243
IFAN$_D$ [41]	28.66	0.868	0.025	0.172	23.46	0.743	0.049	0.240	25.99	0.804	0.037	0.207
Restormer$_D$	**29.48**	**0.895**	**0.023**	**0.134**	**23.97**	**0.773**	**0.047**	**0.175**	**26.66**	**0.833**	**0.035**	**0.155**

- 0.6 dB improvements over the previous best method IFAN
- Compared to the Transformer model Uformer [80], our method provides a substantial gain of 1.01 dB PSNR

EXPERIMENT AND ANALYSIS

Defocus Deblurring

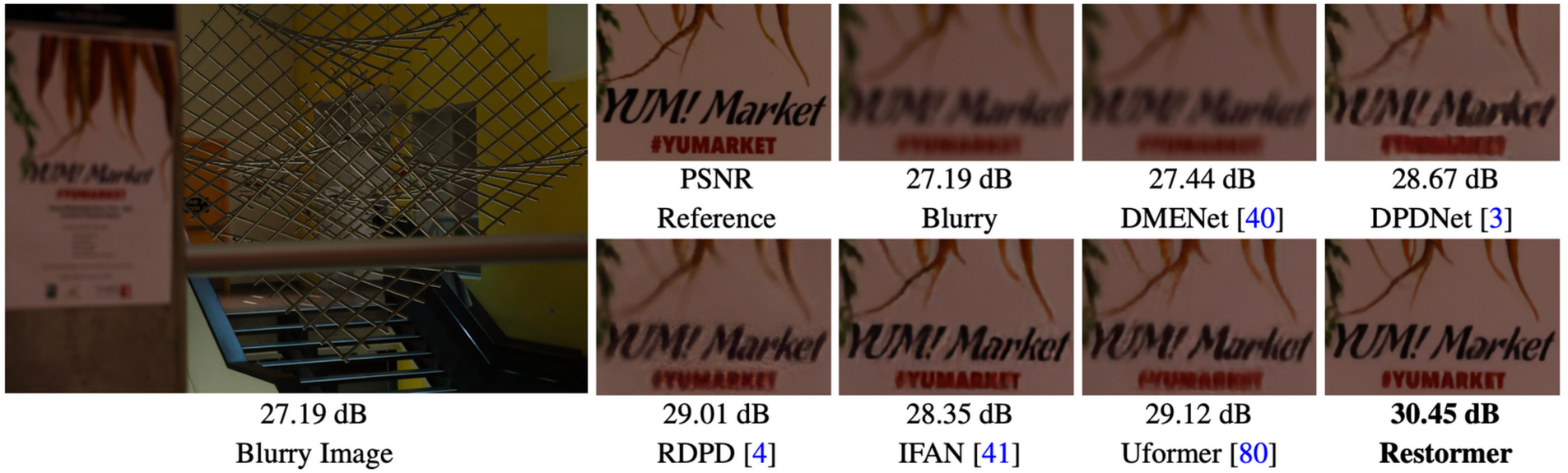


Figure 5. **Dual-pixel defocus deblurring** comparison on the DPDD dataset [3]. Compared to the other approaches, our Restormer more effectively removes blur while preserving the fine image details.

- More effective in removing spa- tially varying defocus blur than other approaches

EXPERIMENT AND ANALYSIS

Image Denoising

Table 6. **Real image denoising** on SIDD [1] and DND [60] datasets. $*$ denotes methods using additional training data. Our Restormer is trained only on the SIDD images and directly tested on DND. Among competing approaches, only Restormer surpasses 40 dB PSNR.

Dataset	Method	DnCNN [101]	BM3D [15]	CBDNet* [25]	RIDNet* [6]	AINDNet* [35]	VDN [89]	SADNet* [12]	DANet+* [90]	CycleISP* [91]	MIRNet [92]	DeamNet* [63]	MPRNet [93]	DAGL [55]	Uformer [80]	**Restormer** (Ours)
SIDD [1]	PSNR ↑	23.66	25.65	30.78	38.71	39.08	39.28	39.46	39.47	39.52	39.72	39.47	39.71	38.94	39.77	**40.02**
	SSIM ↑	0.583	0.685	0.801	0.951	0.954	0.956	0.957	0.957	0.957	0.959	0.957	0.958	0.953	0.959	**0.960**
DND [60]	PSNR ↑	32.43	34.51	38.06	39.26	39.37	39.38	39.59	39.58	39.56	39.88	39.63	39.80	39.77	39.96	**40.03**
	SSIM ↑	0.790	0.851	0.942	0.953	0.951	0.952	0.952	0.955	0.956	0.956	0.953	0.954	0.956	0.956	**0.956**

- Challenging noise level 50 on high-resolution Urban100 dataset, Restormer achieves 0.37 dB gain over the previous best CNN-based method DRUNet
- 0.31 dB boost over the recent transformer-based network SwinIR
-

EXPERIMENT AND ANALYSIS

Image Denoising

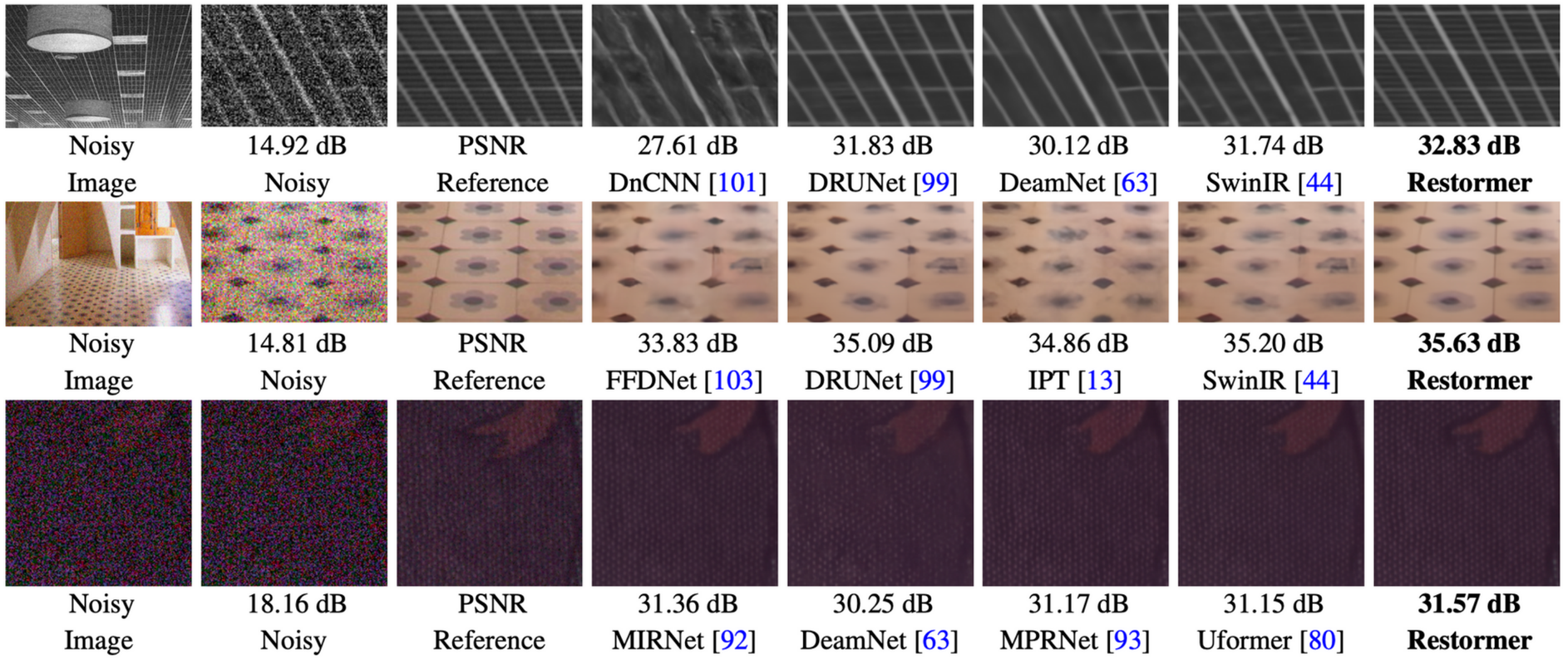


Figure 6. Visual results on **Image denoising**. Top row: Gaussian grayscale denoising. Middle row: Gaussian color denoising. Bottom row: real image denoising. The image reproduction quality of our Restormer is more faithful to the ground-truth than other methods.

EXPERIMENT AND ANALYSIS

Use of Fewer FLOPs

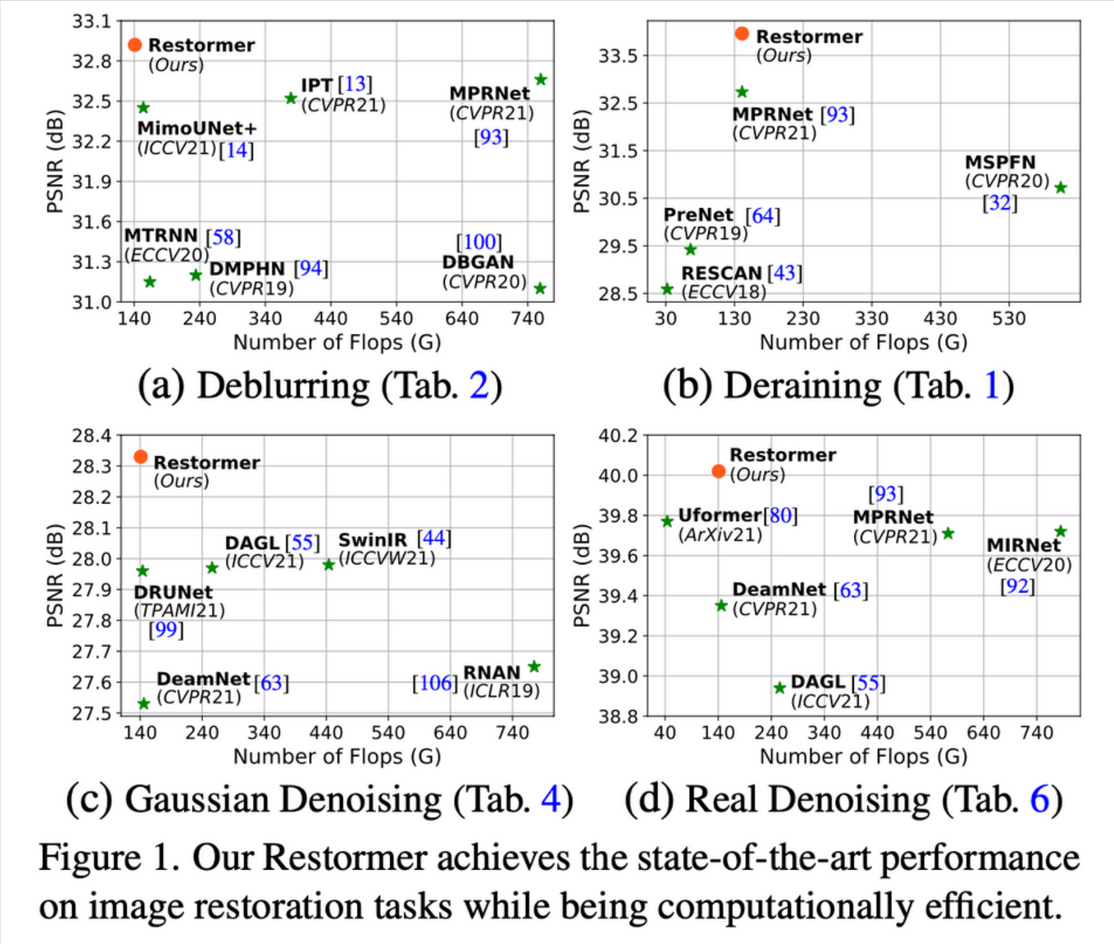


Figure 1. Our Restormer achieves the state-of-the-art performance on image restoration tasks while being computationally efficient.

- Compared to **MPR- Net**, Restormer has 81% fewer FLOPs, which was second best in Motion **Deblurring**
- 0.4 dB improvement over the Transformer model **IPT**, while having 4.4× fewer parameters and runs 29× faster in **Deblurring**
- Compared to **SwinIR** , Restormer has 3.14× fewer FLOPs and runs 13× faster for **Image Denoising**

ABLATION STUDIES

Table 7. **Ablation experiments for the Transformer block.** PSNR is computed on a high-resolution Urban100 dataset [29].

Network		Component	FLOPs (B)	Params (M)	PSNR σ=15	PSNR σ=25	PSNR σ=50
Baseline	**(a)**	UNet with Resblocks [45]	83.4	24.53	34.42	32.18	29.11
Multi-head attention	**(b)**	MTA + FN [77]	83.7	24.84	34.66	32.39	29.28
	(c)	MDTA + FN [77]	85.3	25.02	34.72	32.48	29.43
Feed-forward network	**(d)**	MTA + GFN	83.5	24.79	34.70	32.43	32.40
	(e)	MTA + DFN	85.8	25.08	34.68	32.45	29.42
	(f)	MTA + GDFN	86.2	25.12	34.77	32.56	29.54
Overall	**(g)**	MDTA + GDFN	87.7	25.31	34.82	32.61	29.62

- 7b - Bringing locality to MDTA via depth-wise convolution has improves robustness
- 7c - Demonstrates that MDTA provides favorable gain of 0.32 dB over the baseline
- 7d - Gating mechanism in FN to control information flow yields 0.12 dB gain over the conventional FN
- 7e - Introducing local mechanism to FN also brings performance advantages
- 7f - GDFN achieves PSNR gain of 0.26 dB over the standard FN

ABLATION STUDIES

Table 8. Influence of concat (w/o 1x1 conv) and refinement stage at **decoder (level-1)**. We add the components to experiment Table 7(g).

	PSNR (σ=50)	**FLOPs**	**Params**
Table 7(g)	29.62	87.7	25.31
+ Concat	29.66	110	25.65
+ Refinement	29.71	141	26.12

- Not Using 1×1 convolution after concatenation operation is helpful in preserving fine textural details coming from the encoder
- Demonstrate the effectiveness of adding Transformer blocks in the refinement stage

ABLATION STUDIES

Table 9. Results of training Restormer on fixed patch size and progressively large patch sizes. In **progressive learning** [28], we reduce batch size (as patch size increases) to have similar time per optimization step as of fixed patch training.

Patch Size	**PSNR** (σ=50)	**Train Time (h)**
Fixed (128^2)	29.71	22.5
Progressive (128^2 to 384^2)	29.78	23.1

- Shows that the progressive learning provides better results than the fixed patch training

REFERENCES

[1] Restormer: Efficient Transformer for High-Resolution Image Restoration (Syed Waqas Zamir, Aditya Arora, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, Ming-Hsuan Yang)

[2] https://newsletter.theaiedge.io/p/deep-dive-how-yolo-works-part-1-from

[3]https://www.evidentlyai.com/ranking-metrics/mean-average-precision-map#:~:text=Let's%20try%20to%20add%20some,relevant%20items%20in%20the%20dataset.

[4] PSNR vs SSIM: imperceptibility quality assessment De Rosal Igantius Moses Setiadi]

Thank You!

REFERENCES

Dual Pixel vs Single Pixel