yulunzhang / **RCAN**

PyTorch code for our ECCV 2018 paper "Image Super-Resolution Using Very Deep Residual Channel Attention Networks"

16 commits | **1** branch | **0** releases | **2** contributors

Branch: **master** ▾ | **New pull request** | **Create new file** | **Upload files** | **Find File** | **Clone or download** ▾

yulunzhang Merge pull request #41 from MaFuyan/patch-1 ... Latest commit 51ecfc1 on 19 Mar

Figs	readme	last year
RCAN_TestCode	train_test_code	last year
RCAN_TrainCode	Update __init__.py	4 months ago
README.md	update test path	10 months ago

README.md

Image Super-Resolution Using Very Deep Residual Channel Attention Networks

This repository is for RCAN introduced in the following paper

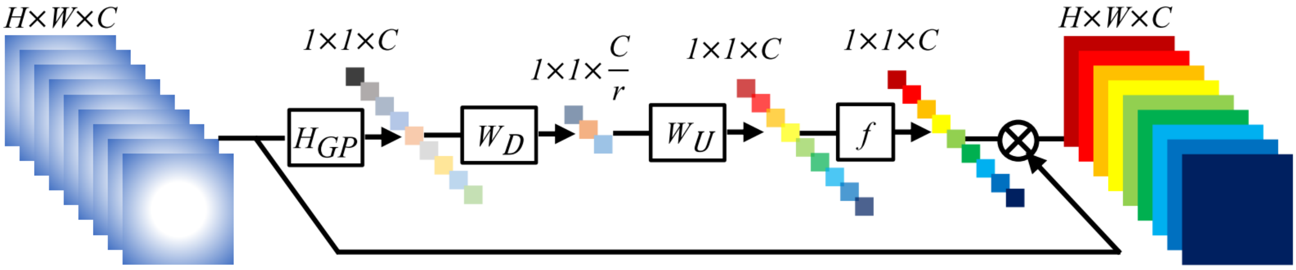
Yulun Zhang, Kunpeng Li, Kai Li, Lichen Wang, Bineng Zhong, and Yun Fu, "Image Super-Resolution Using Very Deep Residual Channel Attention Networks", ECCV 2018, [arXiv]

The code is built on EDSR (PyTorch) and tested on Ubuntu 14.04/16.04 environment (Python3.6, PyTorch_0.4.0, CUDA8.0, cuDNN5.1) with Titan X/1080Ti/Xp GPUs. RCAN model has also been merged into EDSR (PyTorch).

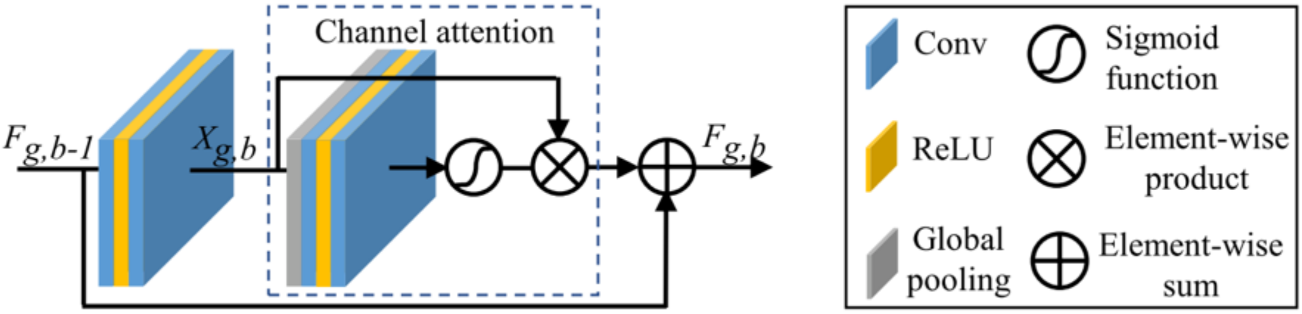
Contents

Introduction

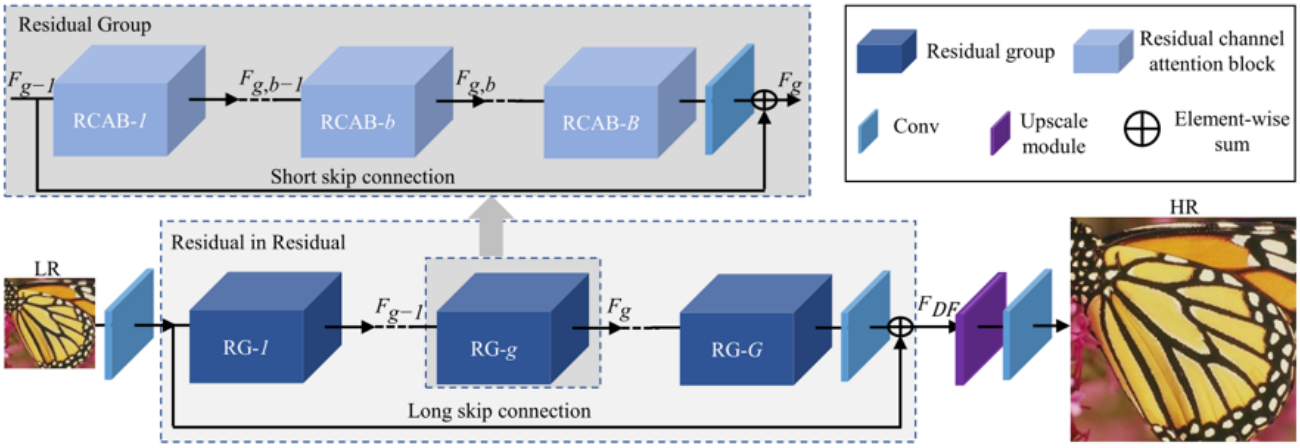
Convolutional neural network (CNN) depth is of crucial importance for image super-resolution (SR). However, we observe that deeper networks for image SR are more difficult to train. The low-resolution inputs and features contain abundant low-frequency information, which is treated equally across channels, hence hindering the representational ability of CNNs. To solve these problems, we propose the very deep residual channel attention networks (RCAN). Specifically, we propose a residual in residual (RIR) structure to form very deep network, which consists of several residual groups with long skip connections. Each residual group contains some residual blocks with short skip connections. Meanwhile, RIR allows abundant low-frequency information to be bypassed through multiple skip connections, making the main network focus on learning high-frequency information. Furthermore, we propose a channel attention mechanism to adaptively rescale channel-wise features by considering interdependencies among channels. Extensive experiments show that our RCAN achieves better accuracy and visual improvements against state-of-the-art methods.



Channel attention (CA) architecture.



Residual channel attention block (RCAB) architecture.



The architecture of our proposed residual channel attention network (RCAN).

Train

Prepare training data

1. Download DIV2K training data (800 training + 100 validtion images) from DIV2K dataset or SNU_CVLab.

2. Specify '--dir_data' based on the HR and LR images path. In option.py, '--ext' is set as 'sep_reset', which first convert .png to .npy. If all the training images (.png) are converted to .npy files, then set '--ext sep' to skip converting files.

For more informaiton, please refer to EDSR(PyTorch).

Begin to train

1. (optional) Download models for our paper and place them in '/RCAN_TrainCode/experiment/model'.

   All the models (BIX2/3/4/8, BDX3) can be downloaded from Dropbox and BaiduYun.

2. Cd to 'RCAN_TrainCode/code', run the following scripts to train models.

   You can use scripts in file 'TrainRCAN_scripts' to train models for our paper.

```
# BI, scale 2, 3, 4, 8
# RCAN_BIX2_G10R20P48, input=48x48, output=96x96
```

```
python main.py --model RCAN --save RCAN_BIX2_G10R20P48 --scale 2 --n_resgroups 10 --n_resblocks 20

# RCAN_BIX3_G10R20P48, input=48x48, output=144x144
python main.py --model RCAN --save RCAN_BIX3_G10R20P48 --scale 3 --n_resgroups 10 --n_resblocks 20

# RCAN_BIX4_G10R20P48, input=48x48, output=192x192
python main.py --model RCAN --save RCAN_BIX4_G10R20P48 --scale 4 --n_resgroups 10 --n_resblocks 20

# RCAN_BIX8_G10R20P48, input=48x48, output=384x384
python main.py --model RCAN --save RCAN_BIX8_G10R20P48 --scale 8 --n_resgroups 10 --n_resblocks 20

# RCAN_BDX3_G10R20P48, input=48x48, output=144x144
# specify '--dir_data' to the path of BD training data
python main.py --model RCAN --save RCAN_BIX3_G10R20P48 --scale 3 --n_resgroups 10 --n_resblocks 20
```

Test

Quick start

1. Download models for our paper and place them in '/RCAN_TestCode/model'.

   All the models (BIX2/3/4/8, BDX3) can be downloaded from Dropbox and BaiduYun.

2. Cd to '/RCAN_TestCode/code', run the following scripts.

   You can use scripts in file 'TestRCAN_scripts' to produce results for our paper.

```
# No self-ensemble: RCAN
# BI degradation model, X2, X3, X4, X8
# RCAN_BIX2
python main.py --data_test MyImage --scale 2 --model RCAN --n_resgroups 10 --n_resblocks 20 --n_fea
# RCAN_BIX3
python main.py --data_test MyImage --scale 3 --model RCAN --n_resgroups 10 --n_resblocks 20 --n_fea
# RCAN_BIX4
python main.py --data_test MyImage --scale 4 --model RCAN --n_resgroups 10 --n_resblocks 20 --n_fea
# RCAN_BIX8
python main.py --data_test MyImage --scale 8 --model RCAN --n_resgroups 10 --n_resblocks 20 --n_fea
# BD degradation model, X3
# RCAN_BDX3
python main.py --data_test MyImage --scale 3 --model RCAN --n_resgroups 10 --n_resblocks 20 --n_fea
# With self-ensemble: RCAN+
# RCANplus_BIX2
python main.py --data_test MyImage --scale 2 --model RCAN --n_resgroups 10 --n_resblocks 20 --n_fea
# RCANplus_BIX3
python main.py --data_test MyImage --scale 3 --model RCAN --n_resgroups 10 --n_resblocks 20 --n_fea
# RCANplus_BIX4
python main.py --data_test MyImage --scale 4 --model RCAN --n_resgroups 10 --n_resblocks 20 --n_fea
# RCANplus_BIX8
python main.py --data_test MyImage --scale 8 --model RCAN --n_resgroups 10 --n_resblocks 20 --n_fea
# BD degradation model, X3
# RCANplus_BDX3
python main.py --data_test MyImage --scale 3 --model RCAN --n_resgroups 10 --n_resblocks 20 --n_fea
```

The whole test pipeline

1. Prepare test data.

   Place the original test sets (e.g., Set5, other test sets are available from GoogleDrive or Baidu) in 'OriginalTestData'.

   Run 'Prepare_TestData_HR_LR.m' in Matlab to generate HR/LR images with different degradation models.

2. Conduct image SR.

See **Quick start**

3. Evaluate the results.

   Run 'Evaluate_PSNR_SSIM.m' to obtain PSNR/SSIM values for paper.

Results

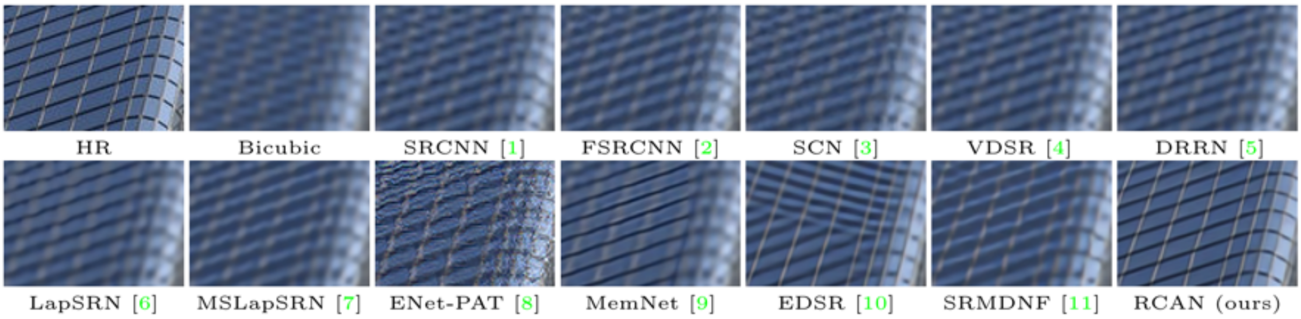
Quantitative Results

Method	Scale	Set5		Set14		B100		Urban100		Manga109	
		PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
Bicubic	×2	33.66	0.9299	30.24	0.8688	29.56	0.8431	26.88	0.8403	30.80	0.9339
SRCNN [1]	×2	36.66	0.9542	32.45	0.9067	31.36	0.8879	29.50	0.8946	35.60	0.9663
FSRCNN [2]	×2	37.05	0.9560	32.66	0.9090	31.53	0.8920	29.88	0.9020	36.67	0.9710
VDSR [4]	×2	37.53	0.9590	33.05	0.9130	31.90	0.8960	30.77	0.9140	37.22	0.9750
LapSRN [6]	×2	37.52	0.9591	33.08	0.9130	31.08	0.8950	30.41	0.9101	37.27	0.9740
MemNet [9]	×2	37.78	0.9597	33.28	0.9142	32.08	0.8978	31.31	0.9195	37.72	0.9740
EDSR [10]	×2	38.11	0.9602	33.92	0.9195	32.32	0.9013	32.93	0.9351	39.10	0.9773
SRMDNF [11]	×2	37.79	0.9601	33.32	0.9159	32.05	0.8985	31.33	0.9204	38.07	0.9761
D-DBPN [16]	×2	38.09	0.9600	33.85	0.9190	32.27	0.9000	32.55	0.9324	38.89	0.9775
RDN [17]	×2	38.24	0.9614	34.01	0.9212	32.34	0.9017	32.89	0.9353	39.18	0.9780
RCAN (ours)	×2	<u>38.27</u>	<u>0.9614</u>	<u>34.12</u>	<u>0.9216</u>	<u>32.41</u>	<u>0.9027</u>	<u>33.34</u>	<u>0.9384</u>	<u>39.44</u>	<u>0.9786</u>
RCAN+ (ours)	×2	**38.33**	**0.9617**	**34.23**	**0.9225**	**32.46**	**0.9031**	**33.54**	**0.9399**	**39.61**	**0.9788**
Bicubic	×3	30.39	0.8682	27.55	0.7742	27.21	0.7385	24.46	0.7349	26.95	0.8556
SRCNN [1]	×3	32.75	0.9090	29.30	0.8215	28.41	0.7863	26.24	0.7989	30.48	0.9117
FSRCNN [2]	×3	33.18	0.9140	29.37	0.8240	28.53	0.7910	26.43	0.8080	31.10	0.9210
VDSR [4]	×3	33.67	0.9210	29.78	0.8320	28.83	0.7990	27.14	0.8290	32.01	0.9340
LapSRN [6]	×3	33.82	0.9227	29.87	0.8320	28.82	0.7980	27.07	0.8280	32.21	0.9350
MemNet [9]	×3	34.09	0.9248	30.00	0.8350	28.96	0.8001	27.56	0.8376	32.51	0.9369
EDSR [10]	×3	34.65	0.9280	30.52	0.8462	29.25	0.8093	28.80	0.8653	34.17	0.9476
SRMDNF [11]	×3	34.12	0.9254	30.04	0.8382	28.97	0.8025	27.57	0.8398	33.00	0.9403
RDN [17]	×3	34.71	0.9296	30.57	0.8468	29.26	0.8093	28.80	0.8653	34.13	0.9484
RCAN (ours)	×3	<u>34.74</u>	<u>0.9299</u>	<u>30.65</u>	<u>0.8482</u>	<u>29.32</u>	<u>0.8111</u>	<u>29.09</u>	<u>0.8702</u>	<u>34.44</u>	<u>0.9499</u>
RCAN+ (ours)	×3	**34.85**	**0.9305**	**30.76**	**0.8494**	**29.39**	**0.8122**	**29.31**	**0.8736**	**34.76**	**0.9513**
Bicubic	×4	28.42	0.8104	26.00	0.7027	25.96	0.6675	23.14	0.6577	24.89	0.7866
SRCNN [1]	×4	30.48	0.8628	27.50	0.7513	26.90	0.7101	24.52	0.7221	27.58	0.8555
FSRCNN [2]	×4	30.72	0.8660	27.61	0.7550	26.98	0.7150	24.62	0.7280	27.90	0.8610
VDSR [4]	×4	31.35	0.8830	28.02	0.7680	27.29	0.0726	25.18	0.7540	28.83	0.8870
LapSRN [6]	×4	31.54	0.8850	28.19	0.7720	27.32	0.7270	25.21	0.7560	29.09	0.8900
MemNet [9]	×4	31.74	0.8893	28.26	0.7723	27.40	0.7281	25.50	0.7630	29.42	0.8942
EDSR [10]	×4	32.46	0.8968	28.80	0.7876	27.71	0.7420	26.64	0.8033	31.02	0.9148
SRMDNF [11]	×4	31.96	0.8925	28.35	0.7787	27.49	0.7337	25.68	0.7731	30.09	0.9024
D-DBPN [16]	×4	32.47	0.8980	28.82	0.7860	27.72	0.7400	26.38	0.7946	30.91	0.9137
RDN [17]	×4	32.47	0.8990	28.81	0.7871	27.72	0.7419	26.61	0.8028	31.00	0.9151
RCAN (ours)	×4	<u>32.63</u>	<u>0.9002</u>	<u>28.87</u>	<u>0.7889</u>	<u>27.77</u>	<u>0.7436</u>	<u>26.82</u>	<u>0.8087</u>	<u>31.22</u>	<u>0.9173</u>
RCAN+ (ours)	×4	**32.73**	**0.9013**	**28.98**	**0.7910**	**27.85**	**0.7455**	**27.10**	**0.8142**	**31.65**	**0.9208**
Bicubic	×8	24.40	0.6580	23.10	0.5660	23.67	0.5480	20.74	0.5160	21.47	0.6500
SRCNN [1]	×8	25.33	0.6900	23.76	0.5910	24.13	0.5660	21.29	0.5440	22.46	0.6950
FSRCNN [2]	×8	20.13	0.5520	19.75	0.4820	24.21	0.5680	21.32	0.5380	22.39	0.6730
SCN [3]	×8	25.59	0.7071	24.02	0.6028	24.30	0.5698	21.52	0.5571	22.68	0.6963
VDSR [4]	×8	25.93	0.7240	24.26	0.6140	24.49	0.5830	21.70	0.5710	23.16	0.7250
LapSRN [6]	×8	26.15	0.7380	24.35	0.6200	24.54	0.5860	21.81	0.5810	23.39	0.7350
MemNet [9]	×8	26.16	0.7414	24.38	0.6199	24.58	0.5842	21.89	0.5825	23.56	0.7387
MSLapSRN [7]	×8	26.34	0.7558	24.57	0.6273	24.65	0.5895	22.06	0.5963	23.90	0.7564
EDSR [10]	×8	26.96	0.7762	24.91	0.6420	24.81	0.5985	22.51	0.6221	24.69	0.7841
D-DBPN [16]	×8	27.21	0.7840	25.13	0.6480	24.88	0.6010	22.73	0.6312	25.14	0.7987
RCAN (ours)	×8	<u>27.31</u>	<u>0.7878</u>	<u>25.23</u>	<u>0.6511</u>	<u>24.98</u>	<u>0.6058</u>	<u>23.00</u>	<u>0.6452</u>	<u>25.24</u>	<u>0.8029</u>
RCAN+ (ours)	×8	**27.47**	**0.7913**	**25.40**	**0.6553**	**25.05**	**0.6077**	**23.22**	**0.6524**	**25.58**	**0.8092**

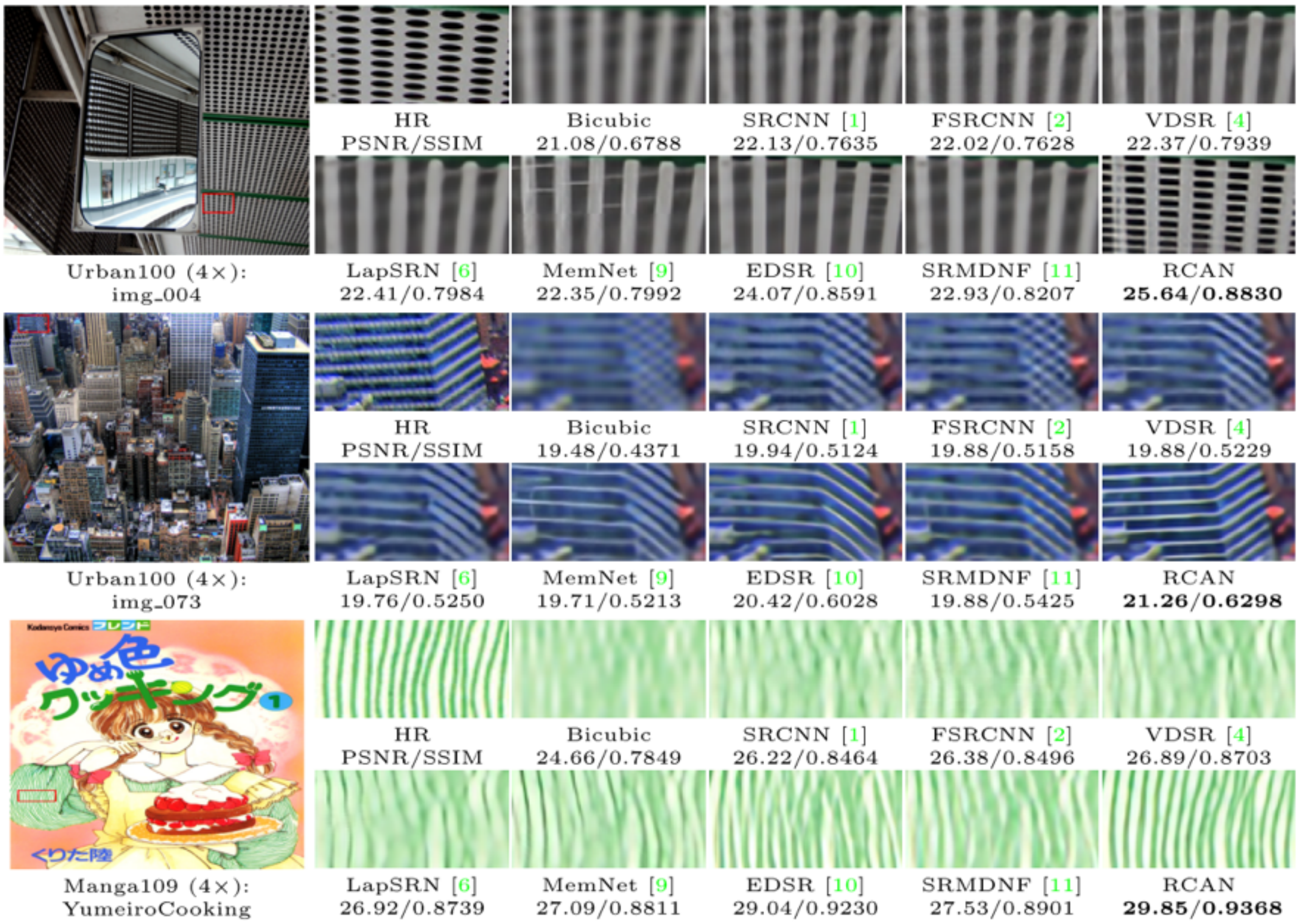
Quantitative results with BI degradation model. Best and second best results are highlighted and underlined

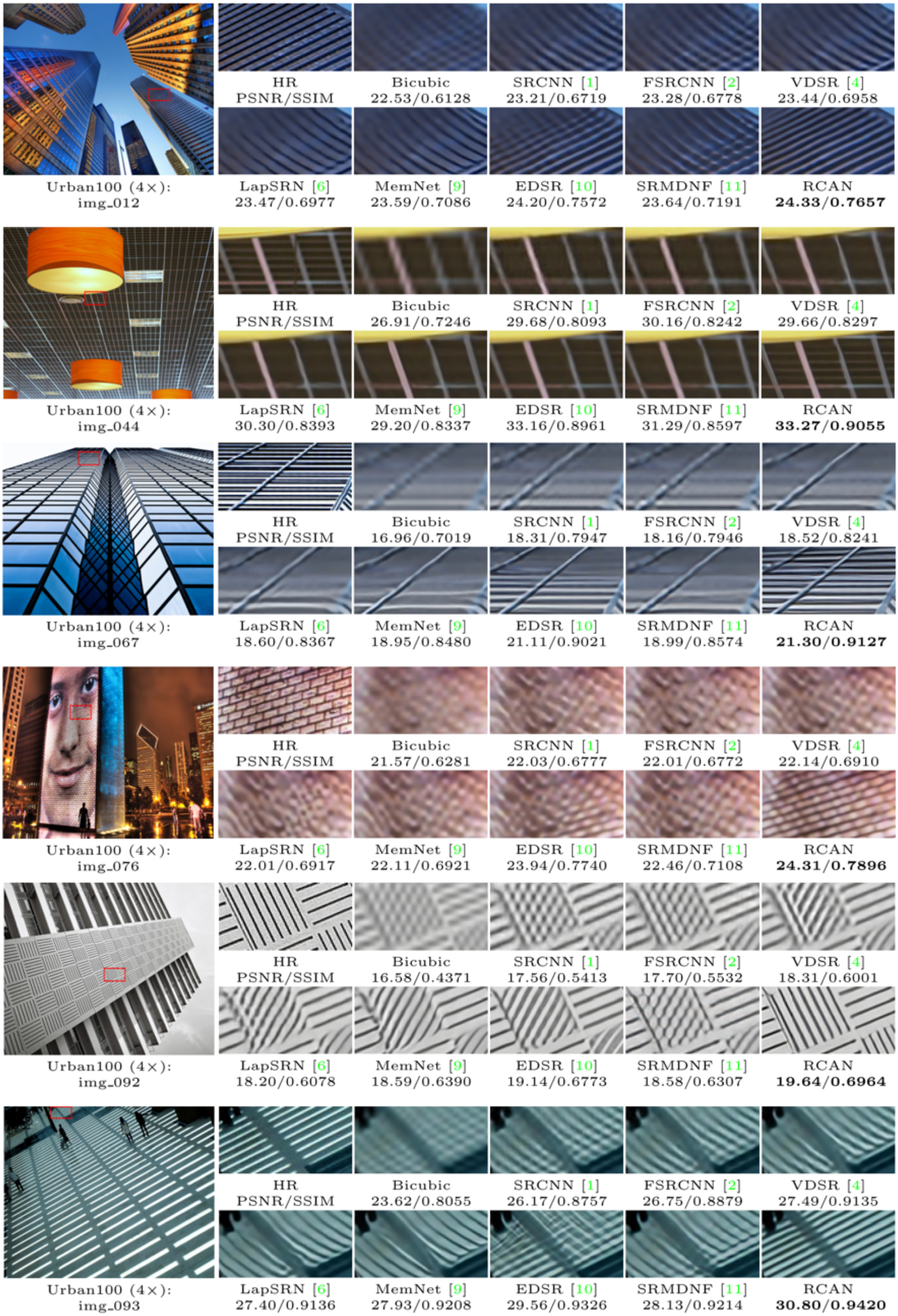
For more results, please refer to our main papar and supplementary file.

Visual Results

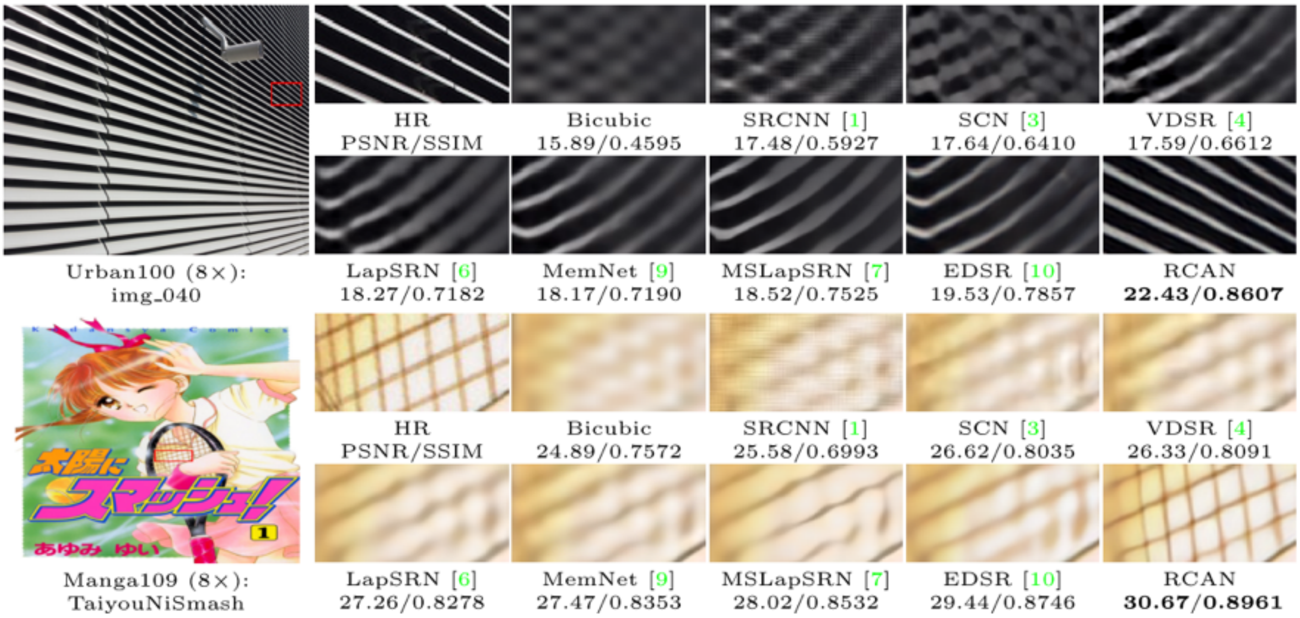


Visual results with Bicubic (BI) degradation (4×) on “img 074” from Urban100

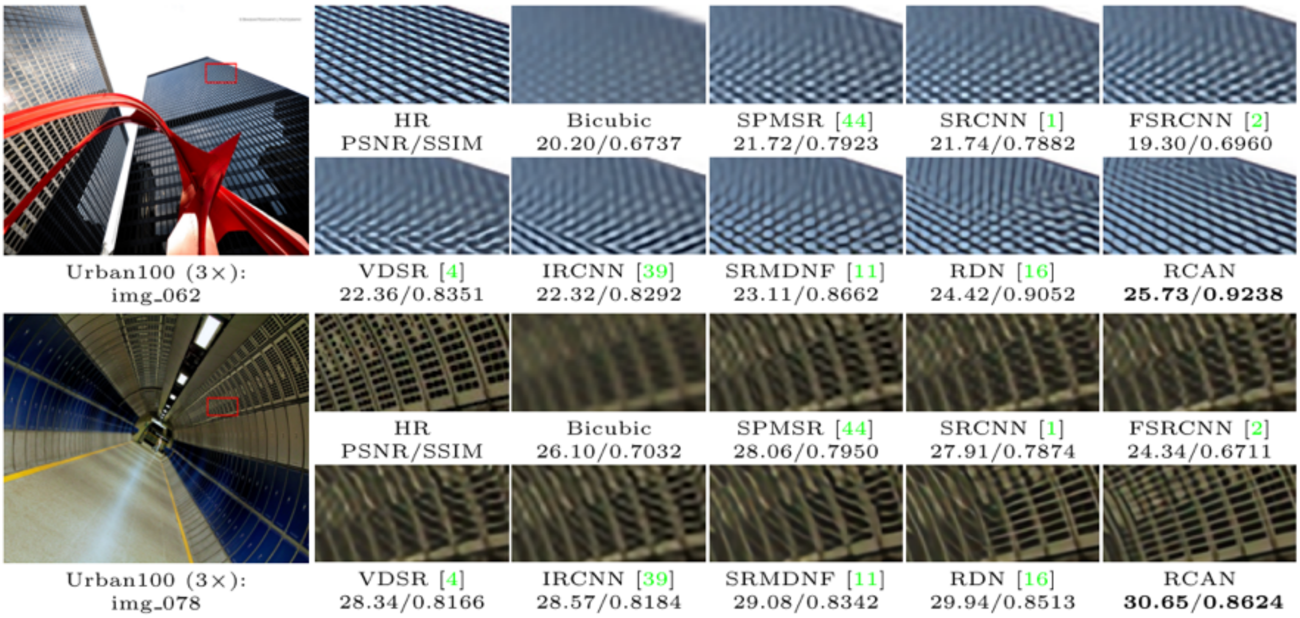




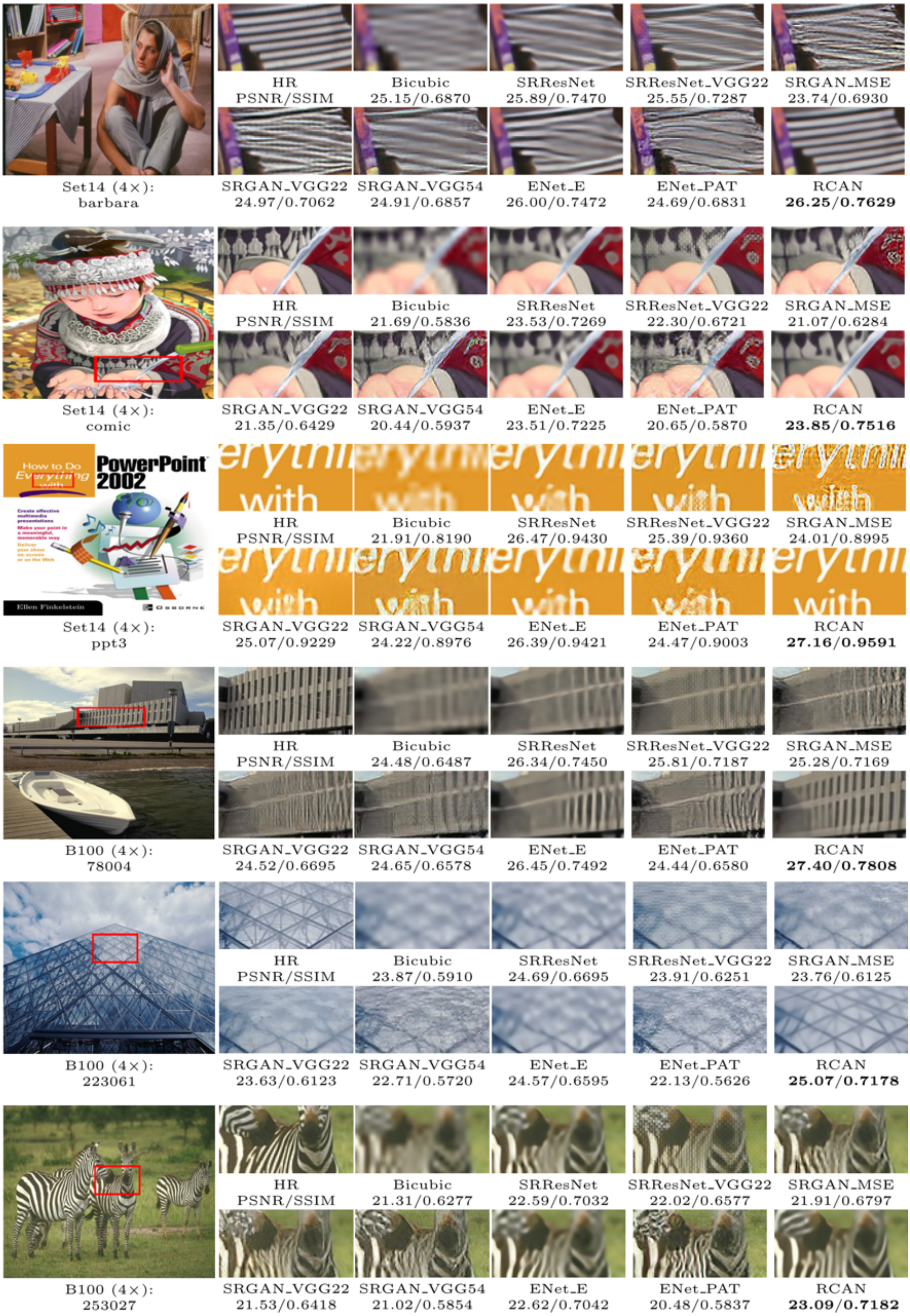
Visual comparison for 4× SR with BI model



Visual comparison for 8× SR with BI model



Visual comparison for 3× SR with BD model



Visual comparison for 4× SR with BI model on Set14 and B100 datasets. The best results are highlighted. SRResNet, SRResNet VGG22, SRGAN MSE, SR- GAN VGG22, and SRGAN VGG54 are proposed in [CVPR2017SRGAN], ENet E and

ENet PAT are proposed in [ICCV2017EnhanceNet]. These comparisons mainly show the effectiveness of our proposed RCAN against GAN based methods

Citation

If you find the code helpful in your resarch or work, please cite the following papers.

```
@InProceedings{Lim_2017_CVPR_Workshops,
  author = {Lim, Bee and Son, Sanghyun and Kim, Heewon and Nah, Seungjun and Lee, Kyoung Mu},
  title = {Enhanced Deep Residual Networks for Single Image Super-Resolution},
  booktitle = {The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops},
  month = {July},
  year = {2017}
}

@inproceedings{zhang2018rcan,
    title={Image Super-Resolution Using Very Deep Residual Channel Attention Networks},
    author={Zhang, Yulun and Li, Kunpeng and Li, Kai and Wang, Lichen and Zhong, Bineng and Fu,
Yun},
    booktitle={ECCV},
    year={2018}
}
```

Acknowledgements

This code is built on EDSR (PyTorch). We thank the authors for sharing their codes of EDSR Torch version and PyTorch version.