Alma Mater Studiorum
University of Bologna

Artificial Intelligence
Machine Learning for Computer Vision

Alessandro Dicosola [ Matr. 935563 ]

# Super Resolution

*Survey Any-Scale Deep Network (ASDN [1])*

# Contents

# 1 Introduction

In computer vision the **super resolution** (hereinafter SR) task is an ill-posed problem for reconstructing a low resolution (hereinafter LR) image to an higher one (hereinafter SR - super resolution image); in order to do so an high resolution image is used (hereinafter HR).

Many method proposed trying to define the images in the features space and apply linear and non-linear operators in order to learn and reconstruct the high resolution image: using random forest, linear and non-linear regressor, manifold embedding[11].

Due to advancement of deep learning, SR task started to be achieved using deep neural networks: from the simplest one, SRCNN[12] with only three convolution for extracting and processing features used then for reconstructing the SR image to more complex ones that use short,long and dense skip connections [13][2] and recursive blocks [14][15], attention mechanism [3][4], meta-learning [7], multi-levels [5][6].

Studies done in deep learning has lead research to understand that deeper and wider networks performs better than shallow and narrow ones as well as using residual learning; skip connections ( short,long and dense ) are used for overcoming the exploding/vanishing gradient problem and at the same time they allow to let networks converge faster; dense connection allow also to reuse features from previous stages and attention mechanism allow networks to focus on most important features (channel-wise and spatial-wise).

SR task achieved with deep learning trying to learn a mapping between LR to HR images given a specific scale. The studies done up to now always used fixed integer scales (2x,3x,4x,. . . ) but none had explored decimal scales but Meta-SR[7] (using meta-learning).

Therefore **Any-Scale Deep Network** (ASDN) tried to explore decimal scale in SR task learning the mapping in an end-to-end way.

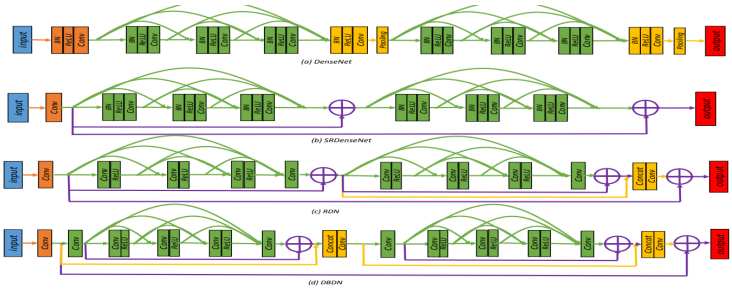


Fig. 2. Simplified structure of (a) DenseNet [26]. The green lines and layers denote the connections and layers in the dense block, and the yellow layers denote the transition and pooling layer. (b) SRDenseNet [15]. The green layers are the same dense block structures as those in DenseNet. The purple lines and element-wise $\bigoplus$ refer to the skip conection. (c) RDN [13]. The yellow lines denote concatenating the blocks output for reconstruction and the green layers are residual dense blocks. (d) DBDN. The yellow lines and layers denote the inter-block dense connection and the green layers are the intra dense blocks. The output goes into upsampling reconstruction layers. The orange layers after input are the feature extraction layers in all models.

Figure 1: Comparison of different network that are using dense connection.

# 2 Background

Here will be explored the most important networks referenced by ASDN [1].

## 2.1 Deep Bi-Dense Network (DBDN[2])

Skip connections allow to highlight one of the most important hyperparameter in deep network: the depth.

But in SR task is important also to use the features learned at the beginning of the network: therefore dense connection were adopted, moreover new architecture arose such as *SRDenseNet* and *RDN*.

All the previous mentioned architectures have a problem: the feature reusing is not really achieved as we can see from the Figure 1 ( the concatenation in the last row use features computed before the sequence of convolutions).

Therefore the aim of Deep Bi-Dense Network (DBDN) is to be able to reuse the feature in the network thanks to dense connections used along the network and within each block.

### 2.1.1 Architectures

We can see from Figure 2a how features are reused: the features extracted by the single feature extraction convolution are processed by a sequence of **Intra dense block**.

The *Intra dense block* contains at the beginning and at the end two convolutions which compress the input and the output channels (to $n_r$) and in the middle $L$ convolutions (with $n_g$ filters) and ReLUs, connected densely, process the compressed input. The compressions are necessary due to the concatenation at the beginning of each intra dense block (Figure 2a) and at the end of each intra dense block (Figure 2b):

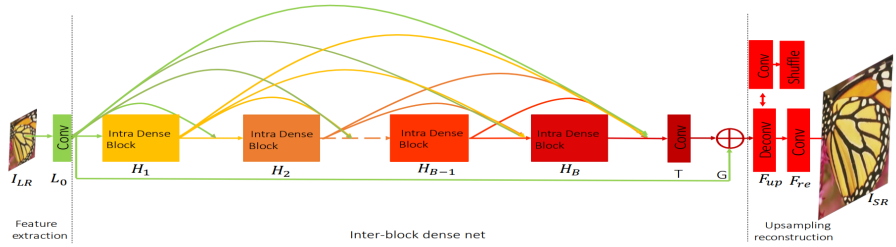



Fig. 3. Network structure

(a) DBDN architecture.

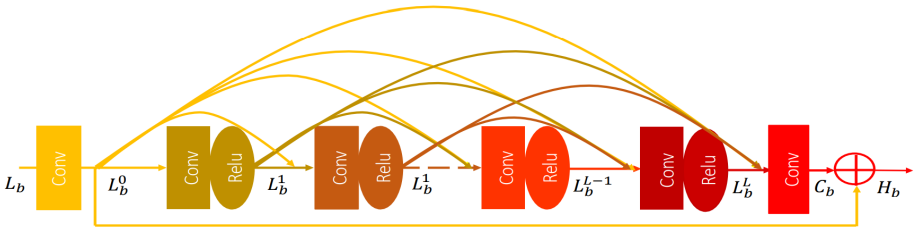



Fig. 4. Intra dense blocks

(b) Intra dense block.

- $n_r *$ number of IDBs before.
- $n_r + L * n_g$ at the end of each IDB.

The local residual learning allow the flowing of the gradients allowing a faster convergence.

Moreover a global skip connections is used in order to use low-frequency information from the LR image and allow the gradient to flow from the end to the beginning of the network.

The SR image is reconstructed using either deconvolution or sub-pixel convolution[10]

### 2.1.2 Results

**Ablation studies on compression layer and inter-block dense connection** From the Figure 3 we can see that *Intra dense-block connection* and *compression layers* are important for achieving good results.

**Quantitative and qualitative results** Thanks to the feature reusing the model perform better than state-of-the-art network maintaining the number of parameters low.

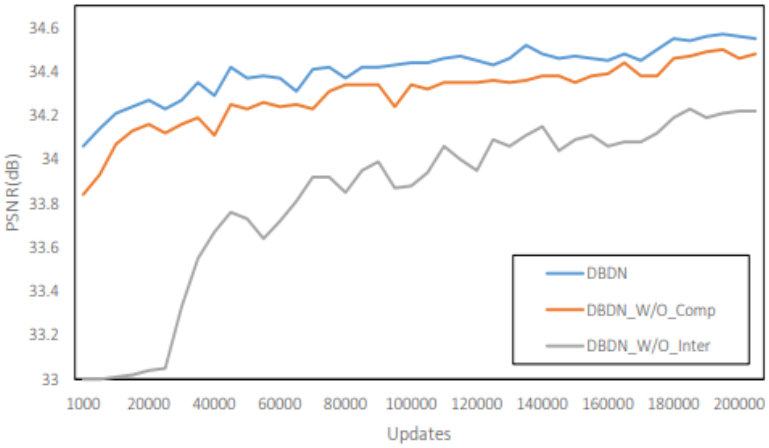



Fig. 8. Discussion about transition layer, skip connection and inter-block dense connectivity in DBDN. The results are evaluated with Set5 for 3× enlargement in 200 epochs

Figure 3: Ablation studies on DBDN.

| Datasets | Scale | Bicubic | VDSR [7] | LapSRN [16] | DRRN [6] | SRDensenet [15] | EDSR [14] | RDN [13] | DBPN [32] | DBDN(ours) | DBDN+(ours) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Set5** | 2× | 33.66/0.9929 | 37.53/0.9587 | 37.52/0.9591 | 37.74/0.9591 | -/- | 38.11/0.9601 | 38.24/0.9614 | 38.09/0.9600 | **38.30/0.9617** | **38.35/0.9618** |
| | 3× | 30.39/0.8682 | 33.66/0.9124 | 33.82/0.9227 | 34.03/0.9244 | -/- | 34.65/0.9282 | 34.71/0.9296 | -/- | **34.76/0.9299** | **34.83/0.9303** |
| | 4× | 28.42/0.8104 | 31.35/0.8838 | 31.54/0.8855 | 31.68/0.8888 | 32.02/0.8934 | 32.46/0.8968 | 32.47/0.8990 | 32.47/0.8980 | **32.54/0.8991** | **32.70/0.9006** |
| **Set14** | 2× | 30.24/0.8688 | 33.03/0.9124 | 33.08/0.9130 | 33.23/0.9136 | -/- | 33.92/0.9195 | 34.01/0.9212 | 33.85/0.9190 | **34.20/0.9224** | **34.34/0.9239** |
| | 3× | 27.55/0.7742 | 29.28/0.8209 | 29.77/0.8314 | 29.96/0.8349 | -/- | 30.52/0.8462 | 30.57/0.8468 | -/- | **30.63/0.8478** | **30.75/0.8495** |
| | 4× | 26.00/0.7027 | 28.01/0.7674 | 28.19/0.7720 | 28.21/0.7721 | 28.50/0.7782 | 28.80/0.7876 | 28.81/0.7871 | 28.82/0.7860 | **28.89/0.7890** | **29.00/0.7908** |
| **BSD100** | 2× | 29.56/0.8431 | 31.90/0.8960 | 31.80/0.8950 | 32.05/0.8973 | -/- | 32.32/0.9013 | 32.34/0.9017 | 32.27/0.9000 | **32.39/0.9022** | **32.45/0.9028** |
| | 3× | 27.21/0.7385 | 28.82/0.7976 | 28.82/0.7973 | 28.95/0.8004 | -/- | 29.25/0.8093 | 29.26/0.8093 | -/- | **29.31/0.8104** | **29.37/0.8112** |
| | 4× | 25.96/0.6675 | 27.29/0.7251 | 27.32/0.7280 | 27.38/0.7284 | 27.53/0.7337 | 27.71/0.7420 | 27.72/0.7418 | 27.72/0.7400 | **27.76/0.7426** | **27.84/0.7446** |
| **Urban100** | 2× | 26.88/0.8403 | 30.76/0.8946 | 30.41/0.9101 | 31.23/0.9188 | -/- | 32.93/0.9351 | 32.84/0.9347 | 32.51/0.9321 | **32.98/0.9364** | **33.36/0.9389** |
| | 3× | 24.46/0.7349 | 26.24/0.7989 | 27.14/0.8272 | 27.53/0.8378 | -/- | 28.80/0.8653 | 28.79/0.8655 | -/- | **28.96/0.8682** | **29.17/0.8715** |
| | 4× | 23.14/0.6577 | 25.18/0.7524 | 25.21/0.7553 | 25.44/0.7638 | 26.05/0.7819 | 26.64/0.8033 | 26.61/0.8028 | 26.38/0.7945 | **26.70/0.8050** | **27.00/0.8117** |
| **Manga109** | 2× | 30.80/0.9339 | 37.16/0.9739 | 37.27/0.9740 | 37.60/0.9736 | -/- | 38.96/0.9769 | 39.18/0.9780 | 38.89/0.9775 | **39.46/0.9788** | **39.65/0.9793** |
| | 3× | 26.95/0.8556 | 31.48/0.9317 | 32.19/0.9334 | 32.42/0.9359 | -/- | 34.17/0.9473 | 34.13/0.9484 | -/- | **34.46/0.9498** | **34.80/0.9512** |
| | 4× | 24.89/0.7866 | 27.82/0.8856 | 29.09/0.8893 | 29.18/0.8914 | 27.83/0.8782 | 31.11/0.9148 | 31.00/0.9151 | 30.91/0.9137 | **31.23/0.9169** | **31.68/0.9198** |

Figure 4: Quantitative results on public benchmark. Red indicates best performance, blue indicates second best performance.


Figure 5: Qualitative results of DBDN.

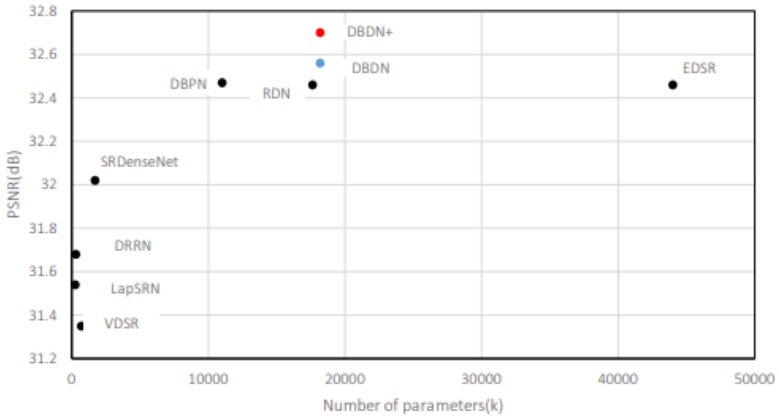



Fig. 7. Performance vs number of parameters. The results are evaluated with Set5 for 4× enlargement. Red indicates the best performance and blue indicates the second best.

Figure 6: Performance-Parameters study.

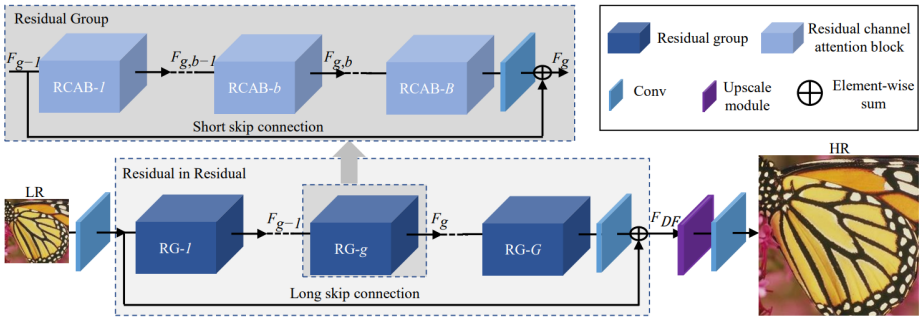



**Fig. 2.** Network architecture of our residual channel attention network (RCAN)

Figure 7: RCAN architecture.

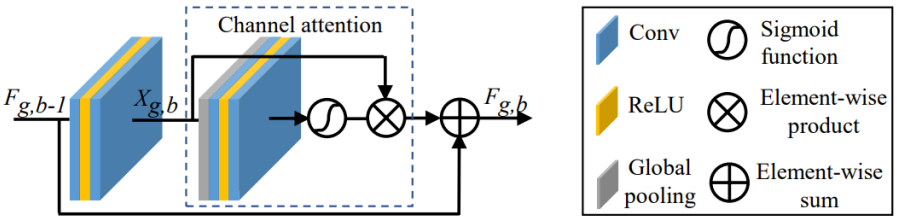



**Fig. 4.** Residual channel attention block (RCAB)

Figure 8: Residual channel attention block.

## 2.2 Residual Channel Attention Network (RCAN[3])

In SR task in order to reconstruct an SR image *high-frequency information* are extremely important, therefore **attention** mechanism are deployed in order to do so.

As usual the depth is also important hence the use of residuals which allow to have a deeper network avoiding exploding/vanishing gradient.

### 2.2.1 Architecture

Figure 7 highlights:

- **shallow extractor** for extracting low level features
- deep features extractor using **Residual in Residual (RIR)**: the main module is the residual groups (**RG**) created using a sequence of **RCAB**[Figure 8] where short skip connection are present which with the long skip connection allow to ease the training and reduce the amount of low-frequency information in the network because directly extracted from the LR image. There are **G** residual groups composed by **B** *residual channel attention*

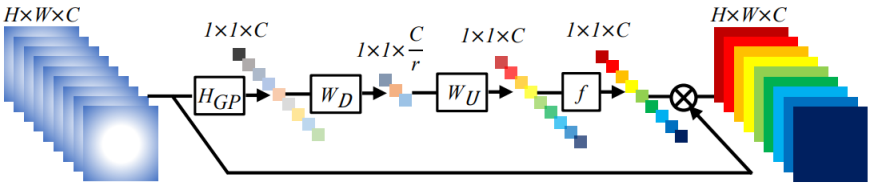



**Fig. 3.** Channel attention (CA). $\otimes$ denotes element-wise product

Figure 9: Channel attention used in RCAB.

*blocks.*

- **upscaling module** using *transposed convolution*, *nearest neighbor and convolution*, *subpixel convolution.*
- **reconstruction** using a single convolution.

**Channel attention** The **Channel attention** has to capture high-frequency information inside the features extracted by convolutions; in order to do so a global statistics of the features is computed using global average pooling channel-wise which are further processed in order to create a mask using a sigmoid activation which allow to create a non mutual-exclusive mask and non-linear information which allow to increase the expressing power of the network.

### 2.2.2 Results

**Ablation studies on LSC, SSC and CA** From Figure 10 it's possible to see that short-skip connections (SSC) and long-skip connections (LSC) are extremely important but in order to improve the final results channel attention is necessary.

**Table 1.** Investigations of RIR (including LSC and SSC) and CA. We observe the best PSNR (dB) values on Set5 (2×) in $5\times10^4$ iterations

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Residual in Residual (RIR) | LSC | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ |
| | SSC | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✓ | ✓ |
| Channel attention (CA) | | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ |
| PSNR on Set5 (2×) | | 37.45 | 37.77 | 37.81 | 37.87 | 37.52 | 37.85 | 37.86 | 37.90 |

Figure 10: Ablation studies done on RCAN.

### Quantitative and qualittive results

**Using Bicubic as degradation model** Thanks to the depth and channel attention RCAN is able to reconstruct the original HR image without artifact or blurring.

**Table 2.** Quantitative results with BI degradation model. Best and second best results are **highlighted** and underlined

| 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 | 38.27 | 0.9614 | 34.12 | 0.9216 | 32.41 | 0.9027 | 33.34 | 0.9384 | 39.44 | 0.9786 |
| 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 | 34.74 | 0.9299 | 30.65 | 0.8482 | 29.32 | 0.8111 | 29.09 | 0.8702 | 34.44 | 0.9499 |
| 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 | 32.63 | 0.9002 | 28.87 | 0.7889 | 27.77 | 0.7436 | 26.82 | 0.8087 | 31.22 | 0.9173 |
| 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 | 27.31 | 0.7878 | 25.23 | 0.6511 | 24.98 | 0.6058 | 23.00 | 0.6452 | 25.24 | 0.8029 |
| RCAN+ (ours) | ×8 | **27.47** | **0.7913** | **25.40** | **0.6553** | **25.05** | **0.6077** | **23.22** | **0.6524** | **25.58** | **0.8092** |

Figure 11: Quantitative results of RCAN.

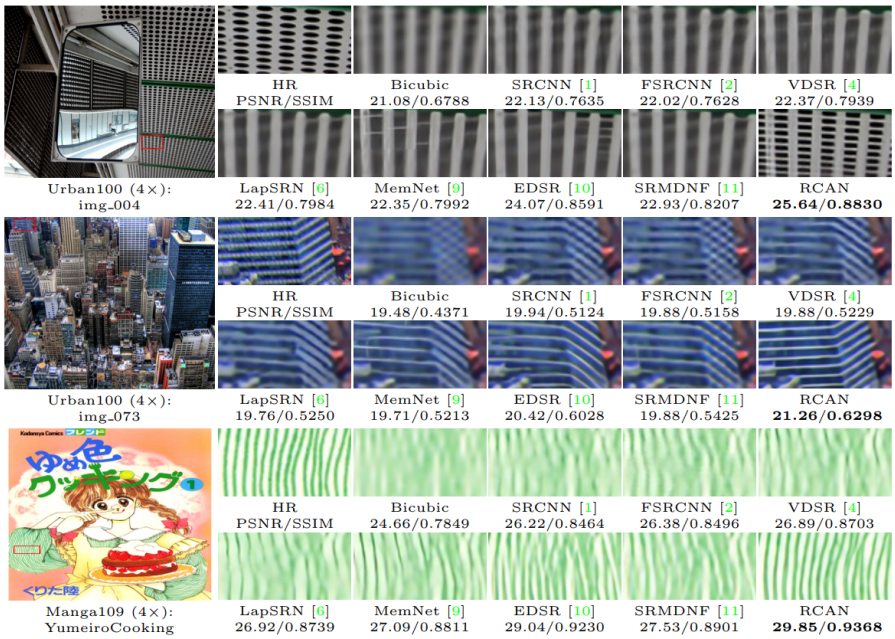



**Fig. 5.** Visual comparison for 4× SR with BI model on Urban100 and Manga109 datasets. The best results are **highlighted**

Figure 12: Qualitative results of RCAN.

**Using Blur-Down degradation model** RCAN performs better than any state-of-the-art.

**Table 3.** Quantitative results with BD degradation model. Best and second best results are **highlighted** and underlined

| Method | Scale | Set5 | | Set14 | | B100 | | Urban100 | | Manga109 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | PSNR | SSIM | PSNR | SSIM | PSNR | SSIM | PSNR | SSIM | PSNR | SSIM |
| Bicubic | ×3 | 28.78 | 0.8308 | 26.38 | 0.7271 | 26.33 | 0.6918 | 23.52 | 0.6862 | 25.46 | 0.8149 |
| SPMSR [44] | ×3 | 32.21 | 0.9001 | 28.89 | 0.8105 | 28.13 | 0.7740 | 25.84 | 0.7856 | 29.64 | 0.9003 |
| SRCNN [1] | ×3 | 32.05 | 0.8944 | 28.80 | 0.8074 | 28.13 | 0.7736 | 25.70 | 0.7770 | 29.47 | 0.8924 |
| FSRCNN [2] | ×3 | 26.23 | 0.8124 | 24.44 | 0.7106 | 24.86 | 0.6832 | 22.04 | 0.6745 | 23.04 | 0.7927 |
| VDSR [4] | ×3 | 33.25 | 0.9150 | 29.46 | 0.8244 | 28.57 | 0.7893 | 26.61 | 0.8136 | 31.06 | 0.9234 |
| IRCNN [15] | ×3 | 33.38 | 0.9182 | 29.63 | 0.8281 | 28.65 | 0.7922 | 26.77 | 0.8154 | 31.15 | 0.9245 |
| SRMDNF [11] | ×3 | 34.01 | 0.9242 | 30.11 | 0.8364 | 28.98 | 0.8009 | 27.50 | 0.8370 | 32.97 | 0.9391 |
| RDN [17] | ×3 | 34.58 | 0.9280 | 30.53 | 0.8447 | 29.23 | 0.8079 | 28.46 | 0.8582 | 33.97 | 0.9465 |
| RCAN (ours) | ×3 | 34.70 | 0.9288 | 30.63 | 0.8462 | 29.32 | 0.8093 | 28.81 | 0.8647 | 34.38 | 0.9483 |
| RCAN+ (ours) | ×3 | **34.83** | **0.9296** | **30.76** | **0.8479** | **29.39** | **0.8106** | **29.04** | **0.8682** | **34.76** | **0.9502** |

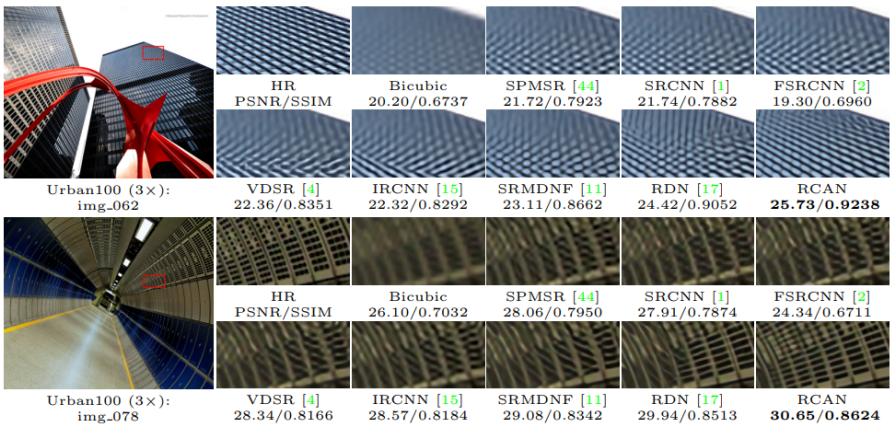



**Fig. 7.** Visual comparison for 3× SR with BD model on Urban100 dataset. The best results are **highlighted**

Figure 13: Quantitative and qualitative results of RCAN using blur as degradation.

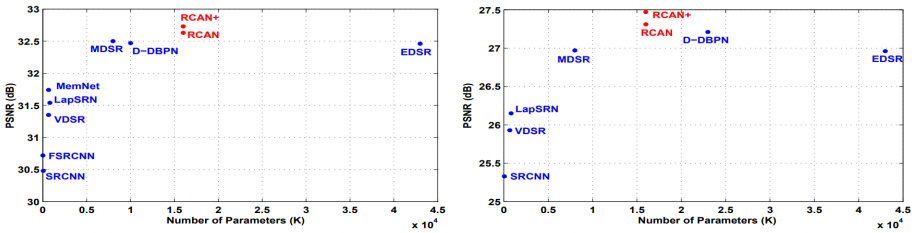
**With object recognition task** Using RCAN for upscaling an 56x56 image to 256x256 achieves the lowest top-1 and top-5 errors in object detection.

**Table 4.** ResNet object recognition performance. The best results are **highlighted**

| Evaluation | Bicubic | DRCN [19] | FSRCNN [2] | PSyCo [45] | ENet-E [8] | RCAN | Baseline |
|---|---|---|---|---|---|---|---|
| Top-1 error | 0.506 | 0.477 | 0.437 | 0.454 | 0.449 | **0.393** | 0.260 |
| Top-5 error | 0.266 | 0.242 | 0.196 | 0.224 | 0.214 | **0.167** | 0.072 |

Figure 14: RCAN used at the beginning of a pipeline (upsamples an image to 224x244) for object recognition

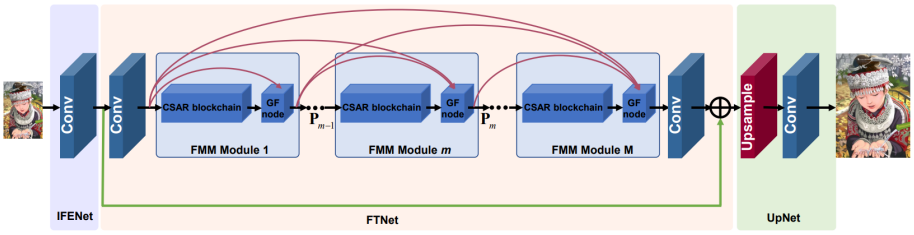
**Performance versus Parameters** RCAN achieve the best performance with not so many parameters.



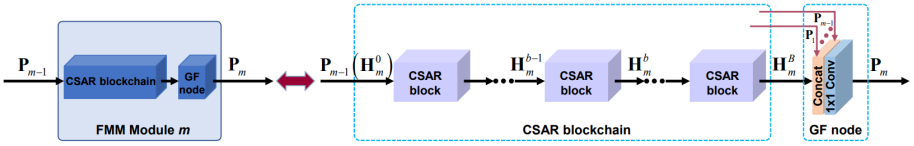

(a) Results on Set5 (4×) (b) Results on Set5 (8×)

**Fig. 8.** Performance and number of parameters. Results are evaluated on Set5

Figure 15: Comparison of parameters and performance of state-of-the-art SR network.




(a) The architecture of the proposed CSFM network




(b) The structure of feature-modulation memory (FMM) module in CSFM network

Fig. 2: The architecture of our CSFM network and the structure of FMM module in CSFM network. (a) The overall architecture of the proposed CSFM network, which adopts adaptive feature-modulation strategy, long-term information persistence mechanism and post-upscaling scheme to boost SR performance. (b) The feature-modulation memory (FMM) module in (a), which exploits a chain of channel-wise and spatial attention residual (CSAR) blocks to capture more informative features and utilizes the gated fusion (GF) node to fusion long-term information from the preceding FMM modules and short-term information from the current module.

Figure 16: CSFM architecture.

## 2.3 Channel-wise and Spatial Feature Modulation Network (CSFM[4])

CSFM exploit features reuse with dense connections, ease the training with skip and dense connections as well as avoid the flow of low-level features inside the network and focus on high-frequency information and on particular spatial position using channel-wise and spatial-wise attention mechanisms.

### 2.3.1 Architecture

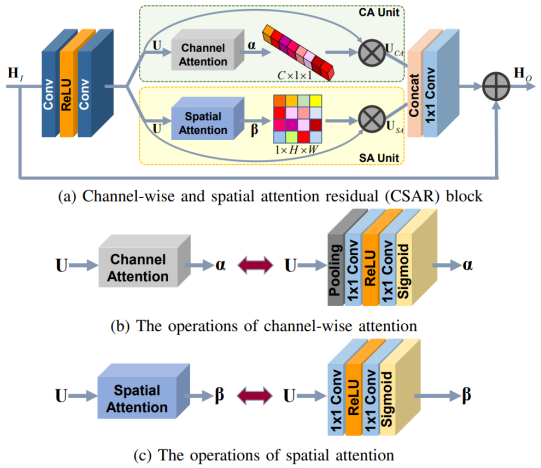
**Feature-Modulation Memory** **CSFM** contains stacked **feature-modulation memory** (*FMM*) which contains a **CSAR blockchain** and a **GF node**.

Since the **gated-function node** (*GF node*) processes not only the features received by the *CSAR blockchain* but also the ones from the output of all previous FMMs, it is preserving long-term features information.

These information flow from one FMM to another but also in the network from shallow levels to deeper ones thanks to the dense connection, this ease the training but also allow to create features at multiple levels.

The *CSAR blockchain* is a sequence of **Channel and spatial attention residual block** (*CSAR*[Figure 17]).

**Channel and spatial attention residual block** **CSAR** is a residual block where there is an identity skip connection that flow low-frequency information, ease the training and preserve shotr-term information and a residual path where




(a) Channel-wise and spatial attention residual (CSAR) block

(b) The operations of channel-wise attention

(c) The operations of spatial attention

Figure 17: CSAR block.

a **channel attention** and **spatial attention** mechanisms are used for extracting high-frequency information feature-wise and spatial-wise.

**CA and SA** The **channel attention**(*CA*) and **spatial attention**(*SA*) are both used for creating a mask in order to improve the expressive power of the network.

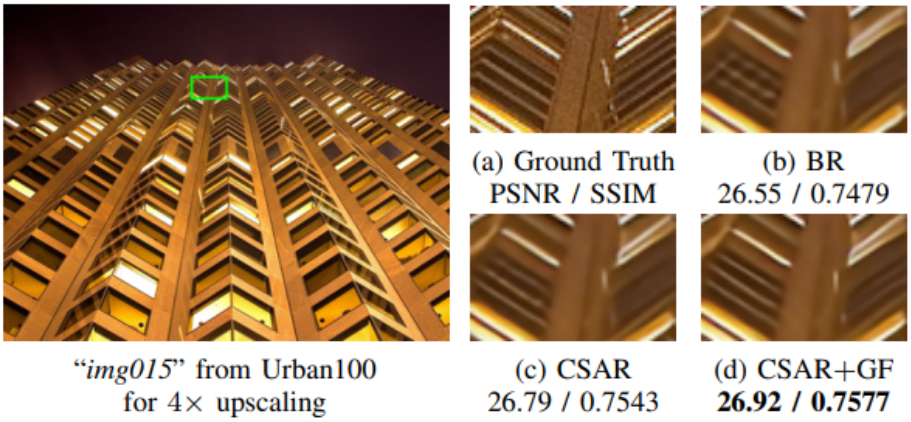
The *CA* extract a channel-wise statistics from the residual processed in *CSAR* using global averaging pooling and processing non-linear features with a sigmoid.

The *SA* extract a spatial-wise statistics simply expanding the channel with the convolution, extracting non-linear features, reducing the channel to the original number and then computing the mask with the same spatial dimension as the input.

### 2.3.2 Results

**Ablation studies** The [Figure 18] have proved the efficacy of the CA, SA (since they are able to increase high-frequency information) and GF (since it's able to process long-term information along the network).

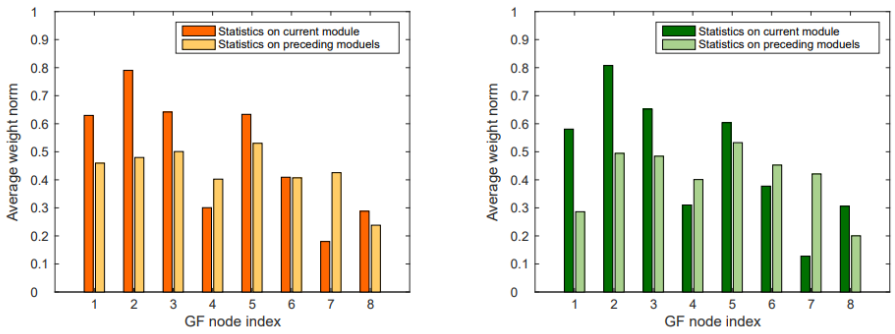
| Components | | Different Combinations of Components | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| In residual blocks | Channel-wise attention (CAR) | × | ✓ | × | – | × | ✓ | × | – |
| | Spatial attention (SAR) | × | × | ✓ | – | × | × | ✓ | – |
| | Combinational attention of CA and SA (CSAR) | × | – | – | ✓ | × | – | – | ✓ |
| Gated fusion (GF) node with long-term dense connections | | × | × | × | × | ✓ | ✓ | ✓ | ✓ |
| PSNR (dB) | | 32.38 | 32.48 | 32.44 | 32.54 | 32.48 | 32.52 | 32.50 | 32.59 |




(a) Results using BR (simple residual block), CSAR(CA and SA), CSAR and GF

Figure 18: Ablation studies for CSFM.

**Analysis of the GF node** The [Figure 19] highlights that long-term information are more important than short-term information (the ones coming from the CSAR blockchain) therefore adding GF is important.




(a) On scale factor 2× (b) On scale factor 4×

Fig. 6: The average weight norms of short-term features from the current FMM module and of long-term features from the preceding FMM modules. (a) The statistics are conducted for a scale factor of 2×. (b) The statistics are conducted for a scale factor of 4×.

Figure 19: Normalized weight norms in eight GF nodes of eight FMM modules for two scale factors of 2× and 4×.

**Number of FMM and CSAR blocks and total amount of parameters** Increasing B and M allow to have a deeper network hence better performance with low cost in number of parameters.

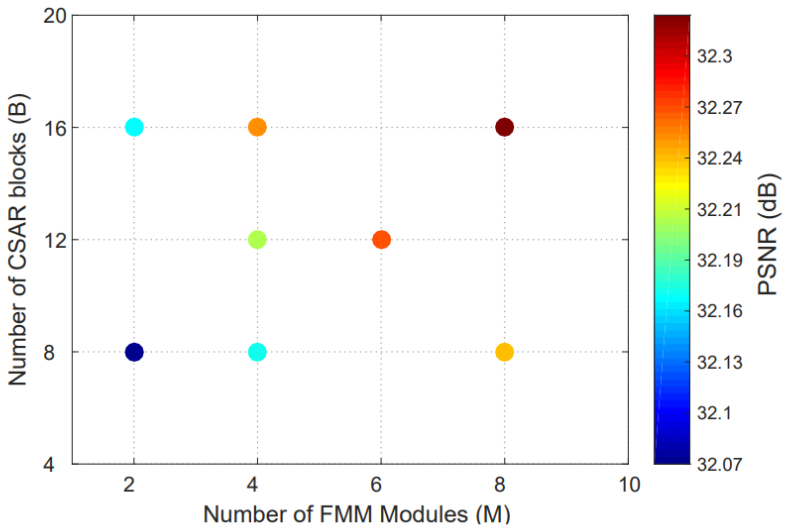



Fig. 7: PSNR performance versus the number of FMM modules (M) and the number of CSAR blocks (B) per FMM . The color of the point denotes the PSNR value that corresponds to the color bar on the right. The tests are conducted for a scale factor of $2\times$ on the dataset of BSD100.

Figure 20: Study done on the number of FMM and CSAR blocks.

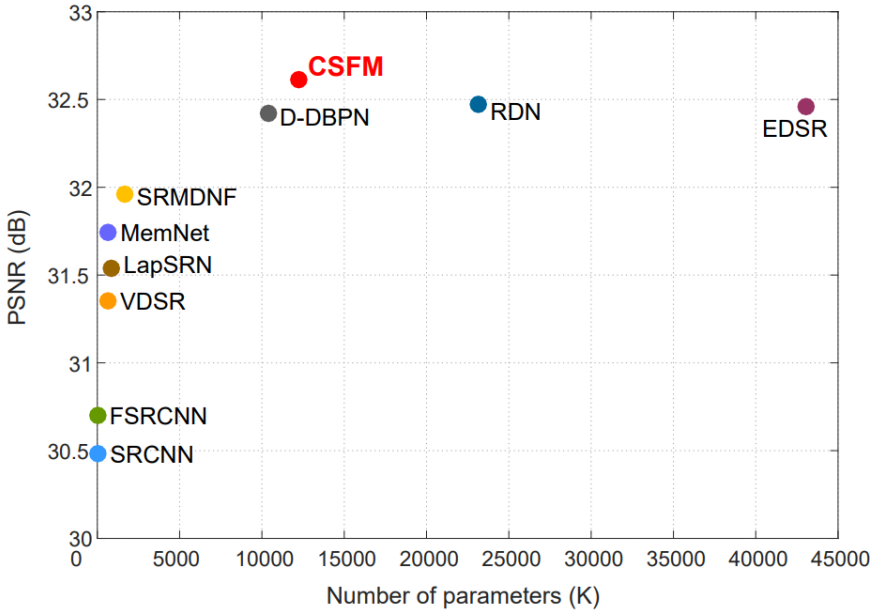



Figure 21: Number of parameters of CSFM with FMM=8 and CSAR=16

**Quantitative and qualitative results** CSFM outperforms all state-of-art-networks.

| Scale | Method | SET5 | | SET14 | | BSD100 | | URBAN100 | | MANGA109 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | PSNR | SSIM | PSNR | SSIM | PSNR | SSIM | PSNR | SSIM | PSNR | SSIM |
| 2× | Bicubic | 33.68 | 0.9304 | 30.24 | 0.8691 | 29.56 | 0.8435 | 26.88 | 0.8405 | 30.81 | 0.9348 |
| | SRCNN [14] | 36.66 | 0.9542 | 32.45 | 0.9067 | 31.36 | 0.8879 | 29.51 | 0.8946 | 35.70 | 0.9677 |
| | FSRCNN [20] | 36.98 | 0.9556 | 32.62 | 0.9087 | 31.50 | 0.8904 | 29.85 | 0.9009 | 36.56 | 0.9703 |
| | VDSR [15] | 37.53 | 0.9587 | 33.05 | 0.9127 | 31.90 | 0.8960 | 30.77 | 0.9141 | 37.41 | 0.9747 |
| | LapSRN [27] | 37.52 | 0.9591 | 32.99 | 0.9124 | 31.80 | 0.8949 | 30.41 | 0.9101 | 37.27 | 0.9740 |
| | DRRN [16] | 37.74 | 0.9591 | 33.23 | 0.9136 | 32.05 | 0.8973 | 31.23 | 0.9188 | 37.88 | 0.9750 |
| | MemNet [19] | 37.78 | 0.9597 | 33.28 | 0.9142 | 32.08 | 0.8978 | 31.31 | 0.9195 | 38.02 | 0.9755 |
| | IDN [31] | 37.83 | 0.9600 | 33.30 | 0.9148 | 32.08 | 0.8985 | 31.27 | 0.9196 | 38.02 | 0.9749 |
| | EDSR [12] | 38.11 | 0.9601 | 33.92 | 0.9195 | 32.32 | 0.9013 | 32.93 | 0.9351 | 39.19 | 0.9782 |
| | SRMDNF [52] | 37.79 | 0.9601 | 33.32 | 0.9154 | 32.05 | 0.8984 | 31.33 | 0.9204 | 38.07 | 0.9761 |
| | D-DBPN [29] | 38.13 | 0.9609 | 33.83 | 0.9201 | 32.28 | 0.9009 | 32.54 | 0.9324 | 38.89 | 0.9775 |
| | RDN [13] | 38.24 | 0.9614 | 34.01 | 0.9212 | 32.34 | 0.9017 | 32.89 | 0.9353 | 39.18 | 0.9780 |
| | CSFM (ours) | **38.26** | **0.9615** | **34.07** | **0.9213** | **32.37** | **0.9021** | **33.12** | **0.9366** | **39.40** | **0.9785** |
| 3× | Bicubic | 30.40 | 0.8686 | 27.54 | 0.7741 | 27.21 | 0.7389 | 24.46 | 0.7349 | 26.95 | 0.8565 |
| | SRCNN [14] | 32.75 | 0.9090 | 29.29 | 0.8215 | 28.41 | 0.7863 | 26.24 | 0.7991 | 30.56 | 0.9125 |
| | FSRCNN [20] | 33.16 | 0.9140 | 29.42 | 0.8242 | 28.52 | 0.7893 | 26.41 | 0.8064 | 31.12 | 0.9196 |
| | VDSR [15] | 33.66 | 0.9213 | 29.78 | 0.8318 | 28.83 | 0.7976 | 27.14 | 0.8279 | 32.13 | 0.9348 |
| | LapSRN [27] | 33.82 | 0.9227 | 29.79 | 0.8320 | 28.82 | 0.7973 | 27.07 | 0.8271 | 32.21 | 0.9344 |
| | DRRN [16] | 34.03 | 0.9244 | 29.96 | 0.8349 | 28.95 | 0.8004 | 27.53 | 0.8378 | 32.74 | 0.9388 |
| | MemNet [19] | 34.09 | 0.9248 | 30.00 | 0.8350 | 28.96 | 0.8001 | 27.56 | 0.8376 | 32.79 | 0.9391 |
| | IDN [31] | 34.11 | 0.9253 | 29.99 | 0.8354 | 28.95 | 0.8013 | 27.42 | 0.8359 | 32.69 | 0.9378 |
| | EDSR [12] | 34.65 | 0.9282 | 30.52 | 0.8462 | 29.25 | 0.8093 | 28.80 | 0.8653 | 34.20 | 0.9486 |
| | SRMDNF [52] | 34.12 | 0.9254 | 30.04 | 0.8371 | 28.97 | 0.8025 | 27.57 | 0.8398 | 33.00 | 0.9403 |
| | RDN [13] | 34.71 | 0.9296 | 30.57 | 0.8468 | 29.26 | 0.8093 | 28.80 | 0.8653 | 34.13 | 0.9484 |
| | CSFM (ours) | **34.76** | **0.9301** | **30.63** | **0.8477** | **29.30** | **0.8105** | **28.98** | **0.8681** | **34.52** | **0.9502** |
| 4× | Bicubic | 28.43 | 0.8109 | 26.00 | 0.7023 | 25.96 | 0.6678 | 23.14 | 0.6574 | 24.89 | 0.7875 |
| | SRCNN [14] | 30.48 | 0.8628 | 27.50 | 0.7513 | 26.90 | 0.7103 | 24.52 | 0.7226 | 27.63 | 0.8553 |
| | FSRCNN [20] | 30.70 | 0.8657 | 27.59 | 0.7535 | 26.96 | 0.7128 | 24.60 | 0.7258 | 27.85 | 0.8557 |
| | VDSR [15] | 31.35 | 0.8838 | 28.02 | 0.7678 | 27.29 | 0.7252 | 25.18 | 0.7525 | 28.87 | 0.8865 |
| | LapSRN [27] | 31.54 | 0.8866 | 28.09 | 0.7694 | 27.32 | 0.7264 | 25.21 | 0.7553 | 29.09 | 0.8893 |
| | DRRN [16] | 31.68 | 0.8888 | 28.21 | 0.7720 | 27.38 | 0.7284 | 25.44 | 0.7638 | 29.45 | 0.8946 |
| | MemNet [19] | 31.74 | 0.8893 | 28.26 | 0.7723 | 27.40 | 0.7281 | 25.50 | 0.7630 | 29.64 | 0.8971 |
| | IDN [31] | 31.82 | 0.8903 | 28.25 | 0.7730 | 27.41 | 0.7297 | 25.41 | 0.7632 | 29.41 | 0.8936 |
| | EDSR [12] | 32.46 | 0.8968 | 28.80 | 0.7876 | 27.71 | 0.7420 | 26.64 | 0.8033 | 31.03 | 0.9158 |
| | SRMDNF [52] | 31.96 | 0.8925 | 28.35 | 0.7772 | 27.49 | 0.7337 | 25.68 | 0.7731 | 30.09 | 0.9024 |
| | D-DBPN [29] | 32.42 | 0.8977 | 28.76 | 0.7862 | 27.68 | 0.7393 | 26.38 | 0.7946 | 30.91 | 0.9137 |
| | RDN [13] | 32.47 | 0.8990 | 28.81 | 0.7871 | 27.72 | 0.7419 | 26.61 | 0.8028 | 31.00 | 0.9151 |
| | CSFM (ours) | **32.61** | **0.9000** | **28.87** | **0.7886** | **27.76** | **0.7432** | **26.78** | **0.8065** | **31.32** | **0.9183** |

Figure 22: Quantitative result of CSFM (FMM=8 and CSAR=16).

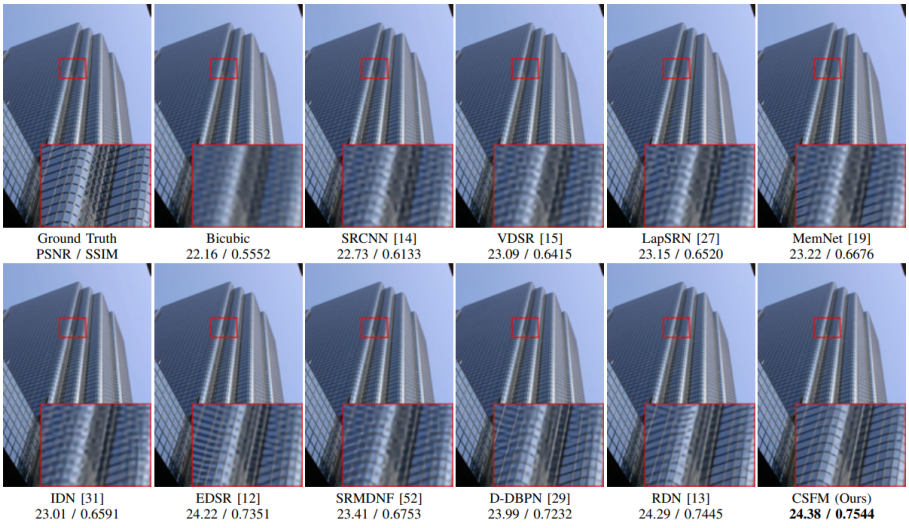



Fig. 12: Visual evaluation for a scale factor of 4× on the image "*img074*" from Urban100. The reconstructed grids produced by our CSFM network are more faithful and sharper than those by other methods.

Figure 23: Qualitative result of CSFM (FMM=8 and CSAR=16).

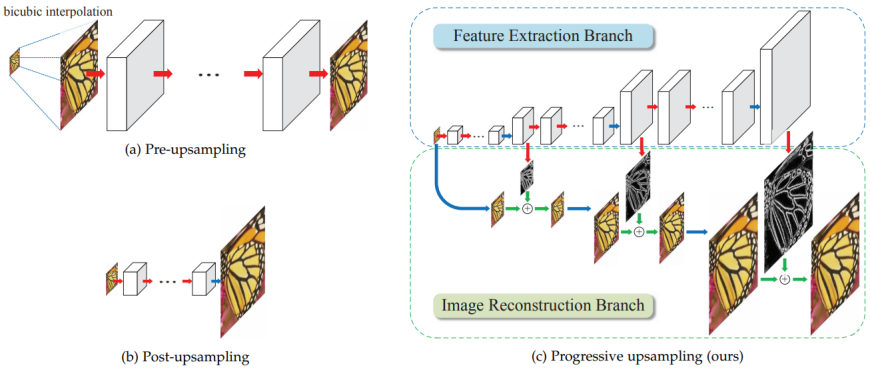



Fig. 1. **Comparisons of upsampling strategies in CNN-based SR algorithms.** Red arrows indicate convolutional layers. Blue arrows indicate transposed convolutions (upsampling), and green arrows denote element-wise addition operators. (a) Pre-upsampling based approaches (e.g., SRCNN [9], VDSR [11], DRCN [12], DRRN [13]) typically use the bicubic interpolation to upscale LR input images to the target spatial resolution before applying deep networks for prediction and reconstruction. (b) Post-upsampling based methods directly extract features from LR input images and use sub-pixel convolution [14] or transposed convolution [15] for upsampling. (c) Progressive upsampling approach using the proposed Laplacian pyramid network reconstructs HR images in a coarse-to-fine manner.

Figure 24: LapSRN architecture.

## 2.4 Deep Laplacian Pyramid Network (LapSRN[5]) and Multi-scale Deep Laplacian Pyramid Network (MS-LapSRN [6])

### 2.4.1 Features

**LapSRN** in order to be able to reconstruct the SR image uses:

- **Charbonnier loss** because the L2 loss is not able to capture the underlying mapping of LR images to many HR images.
- **progressive reconstruction** of the SR image using the Laplacian Pyramid Framework [8].
- **residual learning**: learn the summation between the laplacian extracted by *features extraction branch* and the upscaled LR image in the *image reconstruction branch*

**MS-LapSRN** improve LapSRN introducing:

- **parameter sharing** across pyramid level and within pyramid levels in order to reduce the amount of parameters which increase with the scale (the greater the scale the deeper the network since there are more levels).
- **local skip connections** for avoiding the vanishing/exploding gradient prolem with the increase in the depth of the network.
- **multi scale training**: the previous network was trained for each scale different networks.

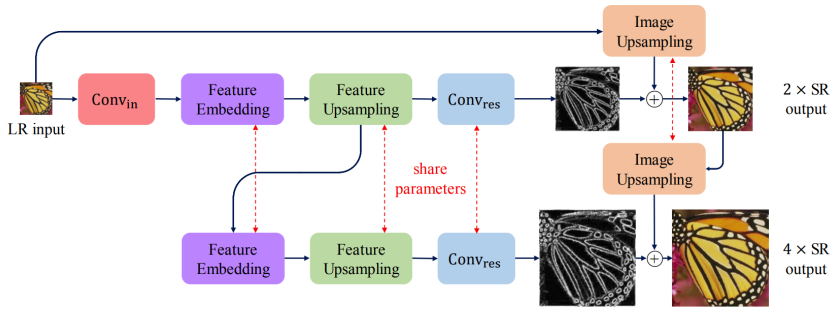



Fig. 3. **Detailed network architecture of the proposed LapSRN.** At each pyramid level, our model consists of a feature embedding sub-network for extracting non-linear features, transposed convolutional layers for upsampling feature maps and images, and a convolutional layer for predicting the sub-band residuals. As the network structure at each level is highly similar, we share the weights of those components across pyramid levels to reduce the number of network parameters.

Figure 25: MS-LapSRN architecture.

### 2.4.2 Architecture

In *LapSRN* [Figure 24] the **feature extraction branch** extract laplacian representations which are used for training (in an end-to-end fashion) the **image reconstruction branch** in order to reconstruct the SR image at the last level (due to the laplacian pyramid framework training on an higher scales lead to have also a network capable to resolve SR task for lower scale).

The residual learning allow the network to focus on high-frequency information (edges) instead of low-frequency ones.

In the *MS-LapSRN* [Figure 25] the feature extractor ($Conv_{in}$, Feature Embedding, Feature Upsampling, $Conv_{res}$) has always the same function: extract meaningful information from an input image in order to create a residual whose spatial dimension in 2x then the input one; therefore is logic to use same weights for doing so.

The *Feature embedding* has **R** recursive block [14] [15] which contains distinct **D** convolutional layers.

The structure of the recursive block [Figure 26b] use a skip connection directly connected to the input and a pre-activation inside the residual path [9].

Recursive block (or single convolution) allow to reduce the number of parameters and increase the depth of the network as well as the receptive field of the network in the last activation without increasing the memory footprint of the network.

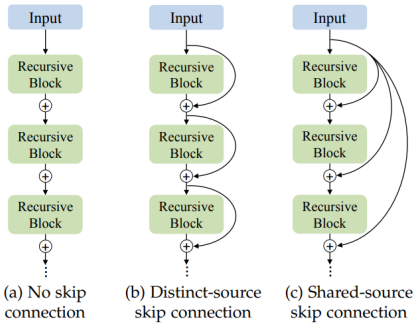



Fig. 4. **Local residual learning.** We explore three different ways of local skip connection in the feature embedding sub-network of the LapSRN for training deeper models.

(a) Differences between local skip connections.

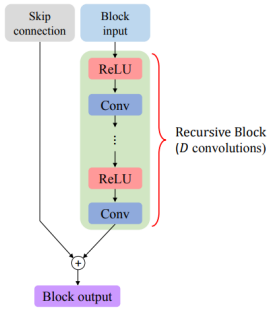



Fig. 5. **Structure of our recursive block.** There are $D$ convolutional layers in a recursive block. The weights of convolutional layers are distinct within the block but shared among all recursive blocks. We use the pre-activation structure [41] without the batch normalization layer.

(b) Recursive block used in MS-LapSRN.

### 2.4.3 Loss

Both *LapSRN* and *MS-LapSRN* use the **Charbonnier loss**: the former use it once the latter at each level; the equation is the following:

$$L_S = \frac{1}{N}\sum_{i=1}^{N}\sum_{l=1}^{L}\rho \times \left(\left(y_l^{(i)} - x_l^{(i)}\right) - r_l^{(i)}\right)$$

where $S$ is the target upsampling factor, $\rho = \sqrt{x^2 + \epsilon^2}$ (Charbonnier penalty function which is a differentiable L1 norm), $x_l$ is the upscaled LR image at level l, $y_l$ is the downscaled HR target image (y) at level l and $r_l$ is the residual at level l.

The final loss is the sum of the loss for each level trained using images with different scales (**multi scales training**):

$$L = \sum_{s\in[2,4,8]} L_s$$

### 2.4.4 Depth

The depth of the network is: $(D \times R + 1) \times (L + 2)$ where $D$ is the number of convolutions inside each recursive block, $R$ is the number of recursive blocks, $+1$ represents the transposed convolution, $+2$ represent the convolutions that perform feature extraction and residual hallucination.

### 2.4.5 Results

Here will be presented only the results of **MS-LapSRN**[6] since it's an evolution of LapSRN.

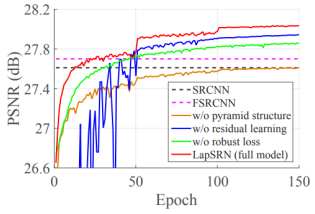



Fig. 6. **Convergence analysis**. We analyze the contributions of the pyramid structures, loss functions, and global residual learning by replacing each component with the one used in existing methods. Our full model converges faster and achieves better performance.

(a) Study on performance with different settings.

TABLE 2
**Ablation study of LapSRN.** Our full model performs favorably against several variants of the LapSRN on both Set5 and Set14 for 4× SR.

| GRL | Pyramid | Loss | Set5 | Set14 |
|---|---|---|---|---|
| ✓ | | Charbonnier | 30.58 | 27.61 |
| | ✓ | Charbonnier | 31.10 | 27.94 |
| ✓ | ✓ | $\mathcal{L}_2$ | 30.93 | 27.86 |
| ✓ | ✓ | Charbonnier | **31.28** | **28.04** |

(b) Ablation study on LapSRN.

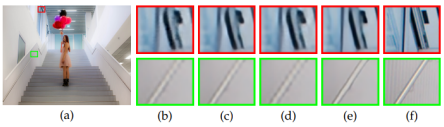



Fig. 7. **Contribution of different components in LapSRN**. (a) Ground truth HR image (b) without pyramid structure (c) without global residual learning (d) without robust loss (e) full model (f) HR patch.

(c) Contribution of different components in LapSRN.


Fig. 8. **Contribution of multi-scale supervision (M.S.)**. The multi-scale supervision guides the network training to progressively reconstruct the HR images and help reduce the spatial aliasing artifacts.

(d) Contribution of multi-scale supervision.

Figure 27: Performance studies done on MS-LapSRN

**Ablation studies on pyramid structure, global residual learning and loss** From the performance studies [Figure 27] it's possible to observe the contribution of each component used for the final result: the pyramid structures allow a faster convergence, the Charbonnier loss a better performance, the global residual learning both.

**Studies on local skip connections** From the local skip connection studies [Figure 28] we can se that the best is *shared source* (**SS**) which use as identity the same input.

**Quantitative evaluation of local residual learning**. We compare three different local residual learning methods on the Urban100 dataset for 4× SR. Overall, the shared local skip connection method ($\text{LapSRN}_{\text{SS}}$) achieves superior performance for deeper models.

| Model | Depth | $\text{LapSRN}_{\text{NS}}$ | $\text{LapSRN}_{\text{DS}}$ | $\text{LapSRN}_{\text{SS}}$ |
|---|---|---|---|---|
| D5R2 | 24 | 25.16 | 25.22 | **25.23** |
| D5R5 | 54 | 25.18 | 25.33 | **25.34** |
| D5R8 | 84 | 25.26 | 25.33 | **25.38** |

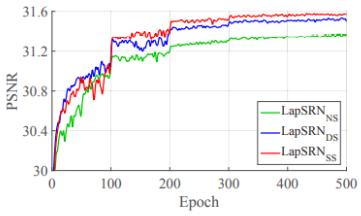



Fig. 9. **Comparisons of local residual learning**. We train our LapSRN-D5R5 model with three different local residual learning methods as described in Section 3.2.3 and evaluate on the Set5 for 4× SR.

Figure 28: Local skip connection studies done on MS-LapSRN

**Parameter sharing in LapSRN.** We reduce the number of network parameters by sharing the weights between pyramid levels and applying recursive layers in the feature embedding sub-network.

| Model | #Parameters | BSDS100 | URBAN100 |
|---|---|---|---|
| LapSRN [16] | 812k | **27.32** | **25.21** |
| $\text{LapSRN}_{\text{NS}}$-D10R1 | 407k | **27.32** | 25.20 |
| $\text{LapSRN}_{\text{NS}}$-D5R2 | 222k | 27.30 | 25.16 |
| $\text{LapSRN}_{\text{NS}}$-D2R5 | 112k | 27.26 | 25.10 |

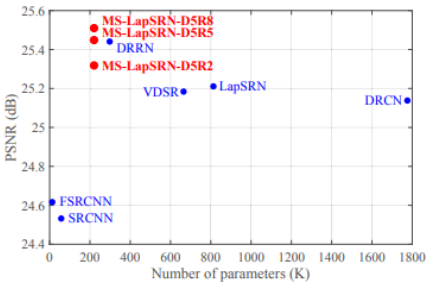



Fig. 15. **Number of network parameters versus performance**. The results are evaluated on the URBAN100 dataset for $4\times$ SR. The proposed MS-LapSRN strides a balance between reconstruction accuracy and execution time.

**Quantitative evaluation of the number of recursive blocks R and the number of convolutional layers D in our feature embedding sub-network**. We build LapSRN with different network depth by varying the values of D and R and evaluate on the BSDS100 and URBAN100 datasets for $4\times$ SR.

| Model | #Parameters | Depth | BSDS100 | URBAN100 |
|---|---|---|---|---|
| D2R5 | 112k | 24 | 27.33 | 25.24 |
| D2R12 | 112k | 52 | 27.35 | **25.31** |
| D2R20 | 112k | 84 | **27.37** | **25.31** |
| D4R3 | 185k | 28 | 27.33 | 25.25 |
| D4R6 | 185k | 52 | **27.37** | 25.34 |
| D4R10 | 185k | 84 | **27.37** | **25.35** |
| D5R2 | 222k | 24 | 27.32 | 25.23 |
| D5R5 | 222k | 54 | 27.38 | 25.34 |
| D5R8 | 222k | 84 | **27.39** | **25.38** |
| D10R1 | 407k | 24 | 27.33 | 25.23 |
| D10R2 | 407k | 44 | 27.36 | 25.27 |
| D10R4 | 407k | 84 | **27.38** | **25.36** |

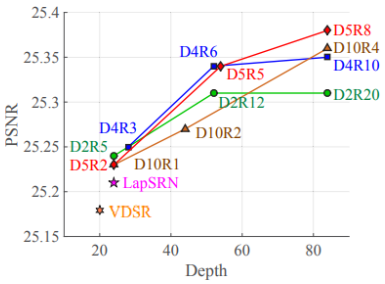
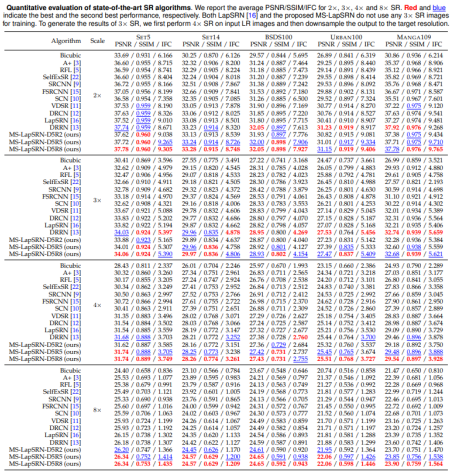



Fig. 10. **PSNR versus network depth**. We test the proposed model with different $D$ and $R$ on the URBAN100 dataset for $4\times$ SR.

Figure 29: Studies on parameters (weights) and hyperparameters D and R on MS-LapSRN



(a) SOtA results up to now

(b) Qualitative results of MS-LapSRN.


(c) Multi-scale results of MS-LapSRN

Figure 30: Quantitative and qualitative results of MS-LapSRN

**Studies on parameter sharing** From the parameters studies [Figure 29] it's possible to see the effectiveness of using recursive block which allow to have a deeper network with an increase of performance without too much parameters and also a faster inference time.

**Quantitative and qualitative results** Fro the quantitative and qualitative results [Figure 30] we can see that overall LapSRN performs better than state-of-the-art network, indeed it is able to reconstruct finer details thanks to the multi-scale training as we can see in the figure.

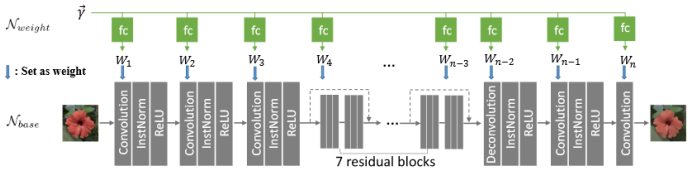



Figure 31: Parametrized Image Operator network.

## 2.5 Magnification-Arbitrary Network (MetaSR [7])

The limitation of SR task using fixed integer scales are:

- real application may be necessary upsampling on continues scales (such as in medical imaging, satellite imaging, ...).
- if the network is trained on a specific scale and is not capable to output intermediate scales (such as [5]) then there are also training cost and memory cost.

In order to overcame fixed integer problems [7] exploit **meta-learning** in order to be able to use decimal scales.

### 2.5.1 Background

The idea is used in [16] where research trained different parametrized operators ( a generic function that transforms an image based on parameters ) using a neural network exploiting meta-learning.

There are two networks Figure 31:

- $N_{base}$ which applies the mapping
- $N_{weights}$ which learns the weights to use inside convolutions in $N_{base}$ based on the parameters $8\vec{\gamma}$) given to the function.

that are learned together end-to-end:

$$L = \|N_{base}(N_{weight}(\vec{\gamma}, I, E)) - f(I, \vec{\gamma})\|_2^2$$

### 2.5.2 Architecture

The **Feature Learning Module** extracts features at low resolution which are used by the **Meta Upscale Module** for generating the SR image.

The *Feature Learning Module* can be any state-of-the-art network while *Meta Upscale Module* replace the upsampling layer in the selected network.

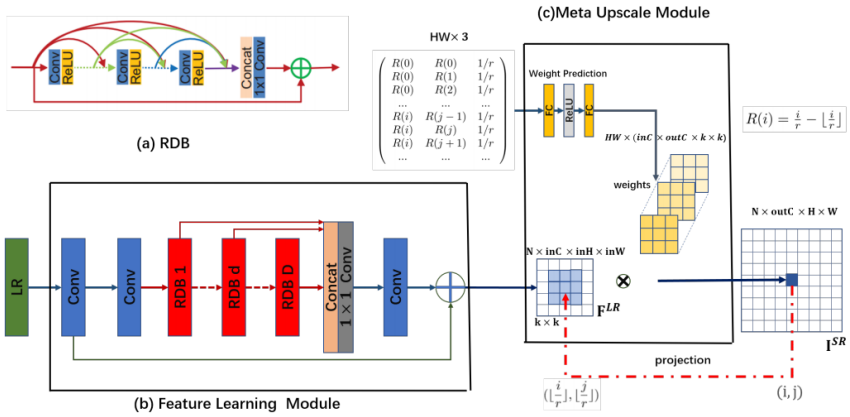



Figure 1. An instance of our Meta-SR based on RDN [36]. We also call the network Meta-RDN. (a) The Residual Dense Block proposed by RDN [36]. (b) The Feature Learning Module which generates the shared feature maps for arbitrary scale factor. (c) For each pixel on the SR image, we project it onto the LR image. The proposed Meta-Upscale Module takes a sequence of coordinate-related and scale-related vectors as input to predict the weights for convolution filters. By doing the convolution operation, our Meta-Upscale finally generate the HR image.

Figure 32: Meta-SR architecture.

**Meta Upscale Module** The **Meta Upscale Module** contains three different modules:

- **location projector** which project pixels from HR images to LR images: $(i', j') = T(i, j) = (\lfloor \frac{i}{r} \rfloor, \lfloor \frac{j}{r} \rfloor)$ where $(i, j)$ are pixels in the HR image and $(i', j')$ are pixels in the LR image.

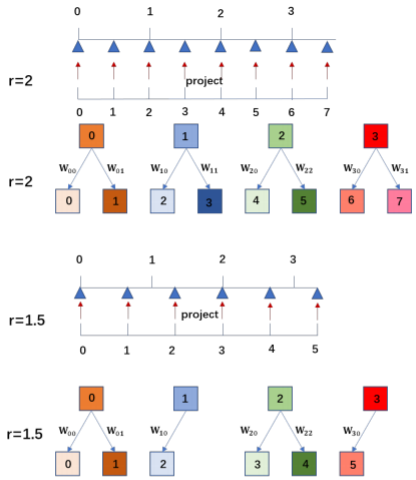



Figure 2. The schematic diagram for how to upscale the feature map with the non-integer scale factor $r = 1.5$. Here we only show the one-dimensional case for simplify.

Figure 33: Location projector: each pixel in the SR image (bottom row) has an unique pixel in the LR image (top row).

- **weight prediction** which predicts the weights of the kernel for each pixel $W(i,j) = \phi(v_{ij}, \theta)$ where $\phi$ is a network with fully connected layers (2 with 256 hidden units ad ReLu as activation).

  $v_{ij}$ is a vector of pixel offset locations:

  $v_{ij} = (\frac{i}{r} - \lfloor \frac{i}{r} \rfloor, \frac{j}{r} - \lfloor \frac{j}{r} \rfloor, \frac{1}{r})$

  (in order to allow the network to distinguish between different scales $\frac{1}{r}$ is inserted in the vector because for example the pixel (i,j) in a 2x image has the same projection and the same weights of the pixels (2i,2j) in a 4x image)

- **feature mapping** which use the features extracted by the *Feature Module* and the weights predicted by the *Weight Prediction* module for reconstructing the HR image:

  $\Phi(F^{LR}(i',j'), W(i,j)) = F^{LR}(i',j') * W(i,j)$ (in practice it does the convolution of the features map with the kernel for getting an intensity or rgb values)

### 2.5.3 Results

**Results with decimal scales** From the results [Figure 34] Meta-SR, and in particular the **Meta upscaling module**, performs better since Meta-RDN

performs better than Meta-Bi (both are learning the weights for upscaling) which perform better than BiConv (where weights are the same for each scale).

The baseline are:

- bicubic
- RDN(x1) and EDSR(x1): LR image is upscaled by $r$ and processed by the network
- RDN(x$k$) and EDSR(x$k$): LR image is processed by the network which apply a upscaling by $k$ then the HR image is downscaled by $\frac{r}{k}$
- BiConv: interpolation of the final feature maps and same convolution for all scales.
- Meta-Bi: interpolation of the final feature maps but use different weights for each scale using the Weight Prediction Network.

| Scale / Methods | X1.1 | X1.2 | X1.3 | X1.4 | X1.5 | X1.6 | X1.7 | X1.8 | X1.9 | X2.0 |
|---|---|---|---|---|---|---|---|---|---|---|
| bicubic | 36.56 | 35.01 | 33.84 | 32.93 | 32.14 | 31.49 | 30.90 | 30.38 | 29.97 | 29.55 |
| RDN(x1) | 42.41 | 39.76 | 38.00 | 36.68 | 35.57 | 34.64 | 33.87 | 33.19 | 32.60 | 32.08 |
| RDN(x2) | 41.84 | 39.34 | 37.87 | 36.63 | 35.56 | 34.63 | 33.83 | 33.1 | 32.52 | 32.11 |
| RDN(x4) | 39.71 | 38.48 | 37.33 | 36.29 | 35.34 | 34.52 | 33.81 | 33.14 | 32.60 | 32.09 |
| BiConv | 41.86 | 39.16 | 37.88 | 29.86 | 35.68 | 34.77 | 33.95 | 33.18 | 32.60 | 31.85 |
| Meta-Bi | 42.11 | 39.58 | 38.07 | 36.83 | 35.81 | 34.86 | 34.03 | 33.24 | 32.63 | 32.18 |
| Meta-RDN(**our**) | **42.82** | **40.40** | **38.28** | **36.95** | **35.86** | **34.90** | **34.13** | **33.45** | **32.86** | **32.35** |
| EDSR(x1) | 42.42 | 39.79 | 38.08 | 36.73 | 35.65 | 34.73 | 33.83 | 33.27 | 32.67 | 32.15 |
| EDSR(x2) | 41.79 | 39.11 | 37.79 | 36.51 | 35.40 | 34.49 | 33.81 | 33.11 | 32.57 | 32.09 |
| EDSR(x4) | 39.61 | 38.41 | 37.27 | 36.24 | 35.30 | 34.46 | 33.75 | 33.09 | 32.56 | 32.04 |
| Meta-EDSR(**our**) | **42.72** | **39.92** | **38.16** | **36.84** | **35.78** | **34.83** | **34.06** | **33.36** | **32.78** | **32.26** |

| Scale / Methods | X2.1 | X2.2 | X2.3 | X2.4 | X2.5 | X2.6 | X2.7 | X2.8 | X2.9 | X3.0 |
|---|---|---|---|---|---|---|---|---|---|---|
| bicubic | 29.18 | 28.87 | 28.57 | 28.31 | 28.13 | 27.89 | 27.66 | 27.51 | 27.31 | 27.19 |
| RDN(x1) | 31.63 | 31.23 | 30.86 | 30.51 | 30.23 | 29.95 | 29.68 | 29.45 | 29.21 | 29.03 |
| RDN(x2) | 31.61 | 31.24 | 30.82 | 30.44 | 30.23 | 29.71 | 29.65 | 29.43 | 29.20 | 29.05 |
| RDN(x4) | 31.61 | 31.23 | 30.88 | 30.52 | 30.31 | 29.99 | 29.75 | 29.53 | 29.26 | 29.14 |
| BiConv | 31.53 | 31.11 | 37.87 | 30.38 | 30.16 | 29.81 | 29.55 | 29.28 | 29.05 | 28.91 |
| Meta-Bi | 31.59 | 31.21 | 30.91 | 30.54 | 30.34 | 30.01 | 29.76 | 29.54 | 29.28 | 29.22 |
| Meta-RDN(**our**) | **31.82** | **31.41** | **31.06** | **30.62** | **30.45** | **30.13** | **29.82** | **29.67** | **29.40** | **29.30** |
| EDSR(x1) | 31.69 | 31.29 | **30.91** | 30.56 | 30.28 | 29.98 | 29.73 | 29.49 | 29.25 | 29.07 |
| EDSR(x2) | 31.57 | 31.15 | 30.81 | 30.47 | 30.22 | 29.91 | 29.66 | 29.45 | 29.19 | 29.09 |
| EDSR(x4) | 31.56 | 31.17 | 30.82 | 30.46 | 30.24 | 29.93 | 29.68 | 29.47 | 29.20 | 29.08 |
| Meta-EDSR(**our**) | **31.73** | **31.31** | 30.87 | **30.60** | **30.40** | **30.09** | **29.83** | **29.61** | **29.34** | **29.22** |

| Scale / Methods | X3.1 | X3.2 | X3.3 | X3.4 | X3.5 | X3.6 | X3.7 | X3.8 | X3.9 | X4.0 |
|---|---|---|---|---|---|---|---|---|---|---|
| bicubic | 26.98 | 26.89 | 26.59 | 26.60 | 26.42 | 26.35 | 26.15 | 26.07 | 26.01 | 25.96 |
| RDN(x1) | 28.81 | 28.67 | 28.47 | 28.30 | 28.15 | 28.00 | 27.86 | 27.72 | 27.59 | 27.47 |
| RDN(x2) | 28.71 | 28.69 | 28.51 | 28.49 | 28.18 | 28.17 | **28.09** | 27.84 | 27.61 | 27.51 |
| RDN(x4) | **28.89** | 28.75 | 28.57 | 28.42 | 28.19 | 28.16 | 27.93 | 27.81 | 27.70 | 27.64 |
| BiConv | 28.64 | 28.51 | 28.28 | 28.13 | 27.91 | 27.84 | 27.61 | 27.49 | 27.37 | 27.29 |
| Meta-Bi | 28.89 | 28.75 | 28.54 | 28.39 | 28.17 | 28.11 | 27.87 | 27.75 | 27.64 | 27.58 |
| Meta-RDN(**our**) | 28.87 | **28.79** | **28.68** | **28.54** | **28.32** | **28.27** | 28.04 | **27.92** | **27.82** | **27.75** |
| EDSR(x1) | 28.85 | 28.69 | 28.51 | 28.36 | 28.18 | 28.04 | 27.91 | 27.77 | 27.64 | 27.52 |
| EDSR(x2) | 28.78 | 28.64 | 28.45 | 28.34 | 28.11 | 28.06 | 27.83 | 27.73 | 27.61 | 27.49 |
| EDSR(x4) | 28.82 | 28.69 | 28.49 | 28.35 | 28.13 | 28.09 | 27.85 | 27.73 | 27.63 | 27.56 |
| Meta-EDSR(**our**) | **28.95** | **28.82** | **28.63** | **28.48** | **28.27** | **28.21** | **27.98** | **27.86** | **27.75** | **27.67** |

Figure 34: Results on Meta-SR

**Comparison between RDN and Meta-RDN** Overall Meta-SR performs better than RDN.


Figure 35: Meta-SR applied for different scales to a single image.

| Methods | Metric | Set14 | | | B100 | | | Manga109 | | | DIV2K | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | X2 | X3 | X4 | X2 | X3 | X4 | X2 | X3 | X4 | X2 | X3 | X4 |
| bicubic | PSNR | 30.24 | 27.55 | 26.00 | 29.56 | 27.21 | 25.96 | 30.80 | 26.95 | 224.89 | 31.35 | 28.49 | 26.92 |
| | SSIM | 0.8688 | 0.7742 | 0.7227 | 0.8431 | 0.7385 | 0.6675 | 0.9339 | 0.8556 | 0.7866 | 0.9076 | 0.8339 | 0.7774 |
| RDN | PSNR | 34.01 | **30.57** | 28.81 | 32.34 | 29.26 | 27.72 | **39.18** | 34.13 | 31.00 | 35.17 | 31.39 | 29.34 |
| | SSIM | 0.9212 | **0.8468** | 0.7871 | 0.9017 | 0.8093 | 0.7419 | 0.9780 | **0.9484** | 0.9151 | 0.9483 | 0.8931 | 0.8446 |
| Meta-RDN | PSNR | **34.04** | 30.55 | **28.84** | **32.35** | **29.30** | **27.75** | 39.18 | **34.14** | **31.03** | **35.18** | **31.42** | **29.36** |
| | SSIM | **0.9213** | 0.8466 | **0.7872** | **0.9019** | **0.8096** | **0.7423** | **0.9782** | 0.9483 | **0.9154** | **0.9484** | **0.8935** | **0.8448** |

Figure 36: Meta-SR versus RDN.

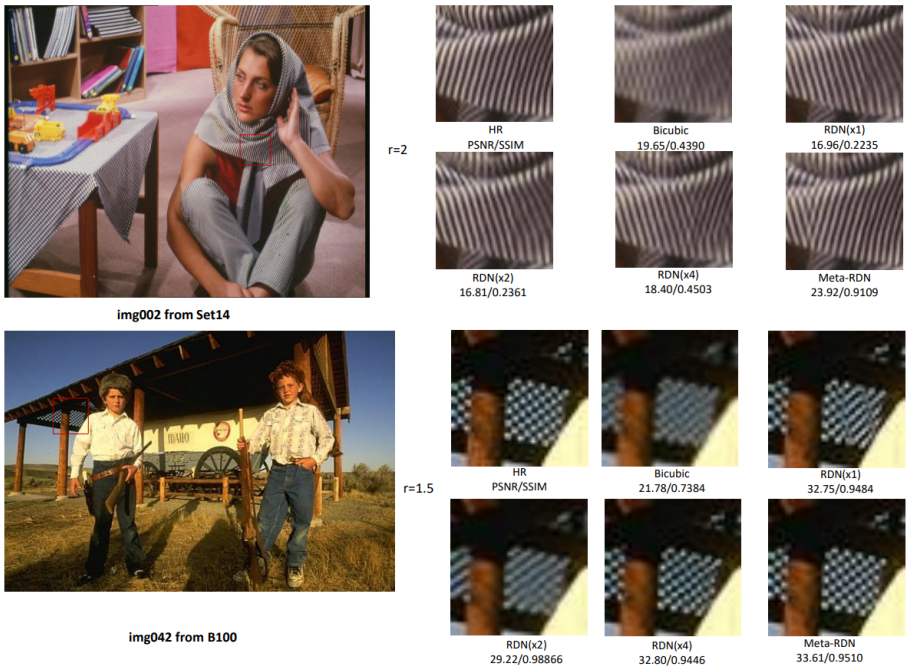



Figure 37: The visual comparison with four baselines. Our MetaRDN has better performance.

## 2.6 Extra

### 2.6.1 The Laplacian Pyramid [8]

In order to compress an image reducing the pixel correlations the *Laplacian Pyramid* was created: subtracting an image with its low-bass band filtered image (using a hand-crafted kernel) it's possible to reduce the variance and the entropy of the image itself; in this way less bits are necessary for representing the image.

The name is due to the similar result obtained applying a laplacian operator to an image.

The kernel used is a 5x5 gaussian-like kernel:

- **separable**: $W(m,n) = \tilde{w}(m) \cdot \tilde{w}(n)$
- **symmetric**: $\tilde{w}(i) = w(-i), i \in [1,2]$, where $w(0) = a$, $w(1) = w(-1) = b$, $w(2) = w(-2) = c$
- **normalized**: $a + 2c + 2b = 1$
- **equal contribution**: $a + 2c = 2b$

which leads to:

$$W = \begin{cases} w(0) = a \\ w(1) = w(-2) = \frac{1}{4} \\ w(2) = w(-2) = \frac{1}{4} - \frac{a}{2} \end{cases}$$

Given $\alpha$ different kernel are created:

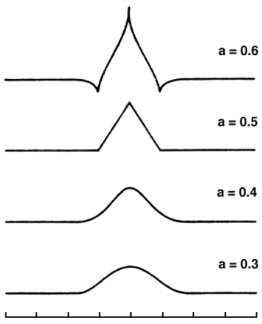



Fig. 3. The shape of the equivalent weighting function depends on the choice of parameter *a*. For *a* = 0.5, the function is triangular; for *a* = 0.4 it is Gaussian-like, and for *a* = 0.3 it is broader than Gaussian. For *a* = 0.6 the function is trimodal.

Figure 38: Kernels as function of $\alpha$

Then, first, a **Gaussian pyramid** is created convolving the kernel with the image *reducing* each time the spatial dimension then each low-bass band filtered image is subtracted with the low-pass band filtered image in the lower level

(after *expanding* it) creating a **Laplacian pyramid** where **high frequency information** are highlighted at different scales.

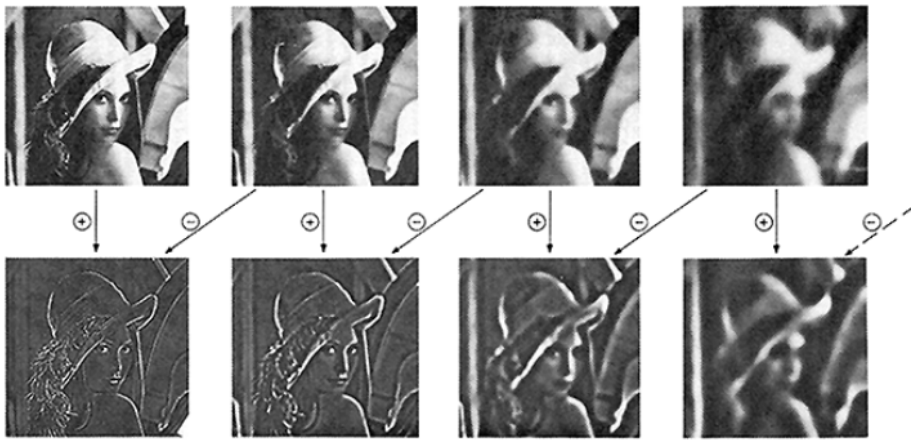


Fig 5. First four levels of the Gaussian and Laplacian pyramid. Gaussian images, upper row, were obtainedby expanding pyramid arrays (Fig. 4) through Gaussian interpolation. Each level of the Laplacian pyramid is the difference between the corresponding and next higher levels of the Gaussian pyramid.

Figure 39: Laplacian pyramid example. The original image is the one on the left. Each image is REDUCED and EXPANDED for creating the Laplacian Pyramid.

The image can be reconstructed starting from the bottom level and expanding and summing two consecutive Laplacian images:

$$g_l = L_l + EXPAND(g_{l+1})$$

Further studies on the entropy and quantization allowed researcher to reduce much more the quantity of information necessary for representing the image with a small loss in quality.

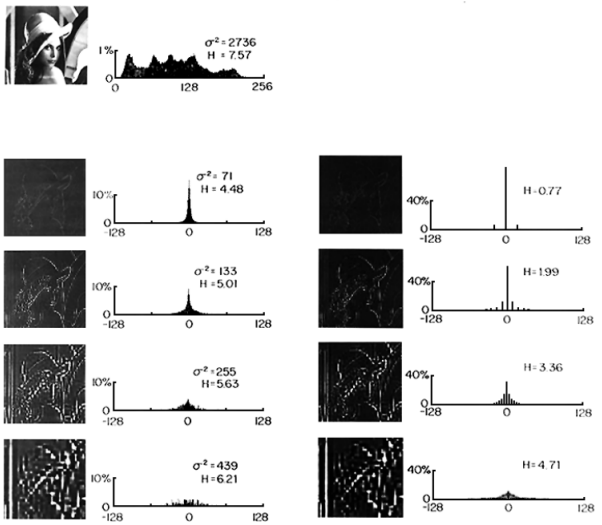



FIG 6. The distribution of pixel gray level values at various stages of the encoding process. The histogram of the original image is given in (a). (b)-(e) give histograms for levels 0-3 of the Laplacian pyramid with generating parameter a=0.6. Histograms following quantization at each level are shown in (f)-(i). Note that pixel values in the Laplacian pyramid are concentrateed near zero, permitting data compression through shortened and varable length code words. Substantial further reduction is realized through quantization (particularly at low pyramid levels) and reduced sample density (particularly at high pyramid levels).

(a) Study on entropy and quantization on the Laplacian Pyramid.

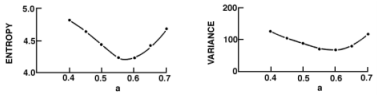



Fig 7. Entropy and variance of pixel values in Laplacian pyramid level 0 as a function of the parameter "$a$" for the "Lady" image. Greatest reduction is obtained for $a \cong 0.6$ This estimate of the optimal "$a$" was also obtained at other pyramid levels and for other images.

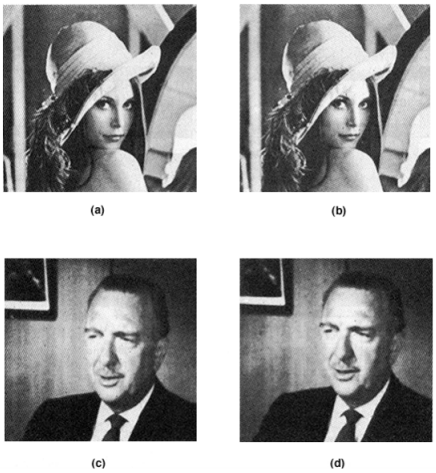



Fig 8. Examples of image data compression using the Laplacian Pyramid code. (a) and (c) give the original "Lady" and "Walter" images, while (b) and (d) give their encoded versions of the data rates are 1.58 and 0.73 bits/pixel for "Lady" and "Walter," respectively. The corresponding mean square errors were 0.88 percent and 0.43 percent, respectively.

(b) Examples with quantization using as width of bins a value such that there is a small degradation on the perception of the image at distance five times greater than the image dimensions.

### 2.6.2 Identity mapping as output of the residual block [9]

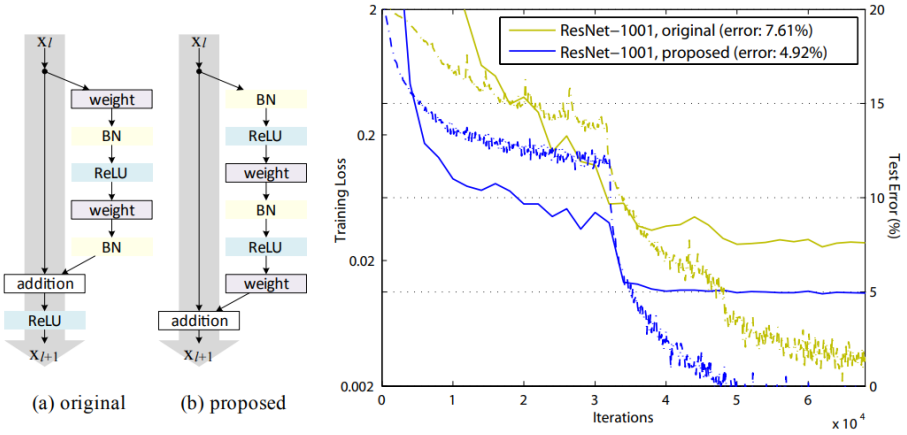



**Figure 1. Left**: (a) original Residual Unit in [1]; (b) proposed Residual Unit. The grey arrows indicate the easiest paths for the information to propagate, corresponding to the additive term "$\mathbf{x}_l$" in Eqn.(4) (forward propagation) and the additive term "1" in Eqn.(5) (backward propagation). **Right**: training curves on CIFAR-10 of **1001-layer** ResNets. Solid lines denote test error (y-axis on the right), and dashed lines denote training loss (y-axis on the left). The proposed unit makes ResNet-1001 easier to train.

Figure 41: The results of using a clean path in the identity.

The result [Figure 41] shows that a **clean information path** from the input and the output (gray line in the figure) allows to ease the training improving the performance.

In order to to so an **identity mapping** should be used in the output of the residual block therefore the researcher proposed a new residual block where the input is **pre-activated**.

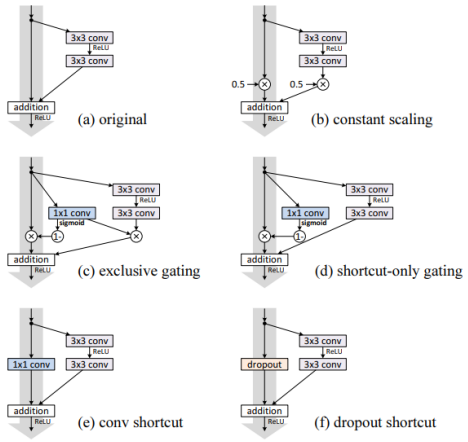
A *Residual unit* can be formalized in this way: $\begin{cases} y_l = h(x_l) + F(x_l, W_l) \\ x_{l+1} = f(y_l) \end{cases}$ where $x_l$ and $x_{l+1}$ are, respectively, the input and the output of the *l-th* residual unit, $y_l$ is the output of the sum between $h$, identity, and $F$ the residual, and f is the $ReLU$.

Using an identity mapping in the output then $x_{l+1} = y_l = x_l + F(x_l, W_l)$ which recursively lead to $X_L = x_l + \sum_i = 1^{L-1} F(x_i, w_i)$: any deeper unit $L$ is the sum of the input of a shallow unit $l$ and the residual of the unit between $l$ and $L$.

An intuition of why this is used in Section 2.4 and DRRN[15] can be seen in this results: since the skip connection allow to flow the real input (the LR image) than the network is able to take apart low frequency information better.

The backpropagation lead to: $\frac{\partial \epsilon}{\partial x_l} = \frac{\partial \epsilon}{\partial x_L}\frac{\partial x_L}{\partial x_l} = \frac{\partial \epsilon}{\partial x_L}\left(1 + \frac{\partial}{\partial x_l}\sum_{i=l}^{L-1} F(x_i, W_i)\right)$ which is never 0 because is unlikely that $\frac{\partial}{\partial x_l}\sum_{i=l}^{L-1} F(x_i, W_i)$ is always $-1$ for all minibatches.

**Studies done on different residual block** In order to prove the mathematical claims they experimented with different combination whose result are worse than the proposed residual block.




(a) The different combination tried.

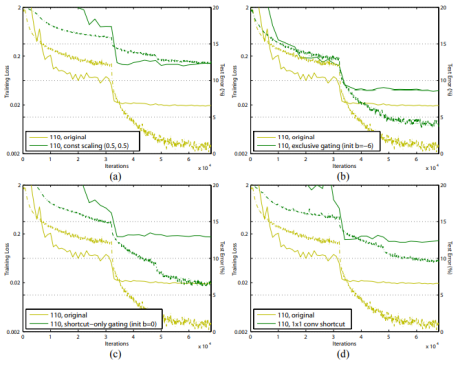



**Figure 3.** Training curves on CIFAR-10 of various shortcuts. Solid lines denote test error (y-axis on the right), and dashed lines denote training loss (y-axis on the left).

(b) The result of the different combination tried.

**Studies on the efffectiveness of the pre-activation** The researchers experimentally prove the correctness of the pre-activation residual block.

**Table 2.** Classification error (%) on the CIFAR-10 test set using different activation functions.

| case | Fig. | ResNet-110 | ResNet-164 |
|---|---|---|---|
| original Residual Unit [1] | Fig. 4(a) | 6.61 | 5.93 |
| BN after addition | Fig. 4(b) | 8.17 | 6.50 |
| ReLU before addition | Fig. 4(c) | 7.84 | 6.14 |
| ReLU-only pre-activation | Fig. 4(d) | 6.71 | 5.91 |
| **full pre-activation** | Fig. 4(e) | **6.37** | **5.46** |

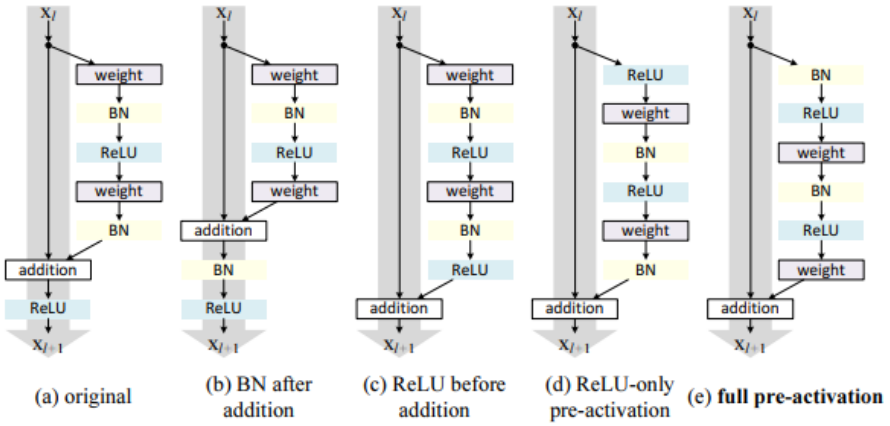



Figure 43: Different combinations of disposition of batch normalization and ReLU

### 2.6.3 Pixel Shuffle: sub-pixel convolution [10]

The **Pixel Shuffle** rearrange the features $C \cdot r^2 \times H \times W$ learned into the LR space into $C \times 2 \cdot H \times 2 \cdot W$ activations learned into the HR space.

In [17], the authors proved that transposed convolution and subpixel convolution return the same result and convolving the features map with a convolution $(r^2 \cdot C_{out},\ C_{in},\ k_H,\ k_W)$ where r is the upscale factor and then rearrange the activations in order to have an activation whose size is $(r \cdot H, r \cdot W)$ is the same as apply a transposed convolution $(C_{out},\ C_{in},\ r \cdot k_H,\ r \cdot k_W)$ ( fractional strided convolution where the input is modified inserting zeros in between and padded in order to have an output upscaled with respect the input)

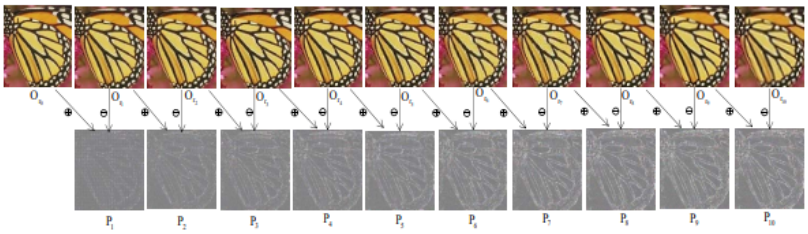


Fig. 2: The Laplacian Frequency Representation has 11 Laplacian pyramid levels ($O_{r_0}, ..., O_{r_{10}}$), with 10 phases in the scale range of $(1, 2]$ ($P_1, ..., P_{10}$). Each phase represents the difference between the successive Laplacian pyramid levels.

Figure 44: Laplacian Frequency Representation using decimal scales

# 3 Any-Scale Deep Network (ASDN[1])

**ASDN** and *Meta-SR[7]*[Section 2.5] are few among SR networks whose objective is to achieve SR images using decimal scales.

In order to do so *Meta-SR* used a *meta-upscaling module* which learns the weights used for convolving features extracted by a LR image given the scale; therefore *Meta-SR* is able to predict the SR image only for scales used for training.

For overcoming this and the impossibility to train a network for all possible decimal scales, *ASDN* eases the task using a **Laplacian Frequency Representation**: exploiting the Laplacian Framework with decimal scales (seen in Section 2.4) the SR image is reconstructed interpolating two nearest neighbors of the decimal scale in the laplacian pyramid.

## 3.1 Laplacian Frequency Representation

Each level learns the high frequency image at the specific scale which is defined as:

$$r_l = \frac{l}{L-1} + 1$$

then given any decimal scale **r** the nearest neighbor is chosen with:

$$i = \lceil (L-1) * (r-1) \rceil$$

and the weights used for interpolating are:

$$w_r = (L-1) * (r_i - r)$$

therefore the **SR image** is:

$$O_r = O_{r_i} + w_r * P_i$$

where $P_i = O_{r_{i-1}} - O_{r_i}$

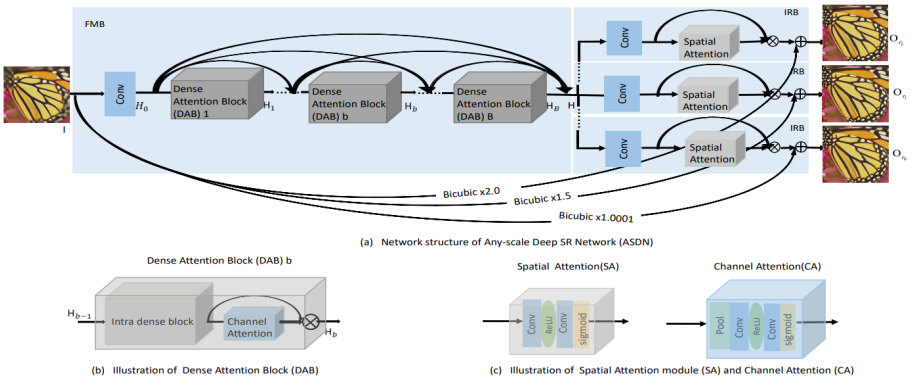



Figure 45: ASDN architecture

The pyramid is named *Laplacian Frequency Representation* (*LFR*) because at each level a phase is computed which represents the high frequency information of the input image as if we are applying a laplacian operator, indeed looking at Figure 44 what we have is a sort of edge detector.

## 3.2 Recursive deployment

In order to avoid to train the network for all possible decimal scales the *Laplacian Frequency Representation* is fixed to a range therefore in order to be able to upscale decimal scales greater thant the greatest scale in the pyramid the network is deployed recursively. In general a large scale $R$ can be defined as $R = r^N$ where $N$ is the number of recursions.

The best scales range is found to be $(1, 2]$ therefore the network should be deployed $N - 1$ times with $r = 2$ and once with $r = \frac{R}{2^{N-1}}$ where $N = \lceil \log_2 R \rceil$

## 3.3 Architecture

Looking at [Figure 45], ASDN has:

- a shallow features extractor
- a deeper features extractor, **Feature Mapping Branch** (*FMB*)
- at each level in the *LFR* the features are reconstructed using the **Image Reconstruction Branch** (*IRB*)
- the *FMB* is shared among all *IRBs*

The *ASDN* is the aggregator of all previous studies:

- the **bi-dense connections** allow to reuse all features within the network, maintain long-term information and ease the training avoiding vanishing/exploding gradient (seen in Section 2.1)

- the IDBs taken directly from Section 2.1 allow to maintain short-term information focusing the training on most important features thanks ot the **channel attention**(as seen in Section 2.2 and Section 2.3)
- the IRB project the features in the image space using a 1x1 convolution with $C = 3$ (or 1 in case of gray-scale images) and then the reconstruction is focused on particular region where high-frequency information are laying due to the **spatial attention** mechanism (as seen in Section 2.3)

## 3.4 FSDN

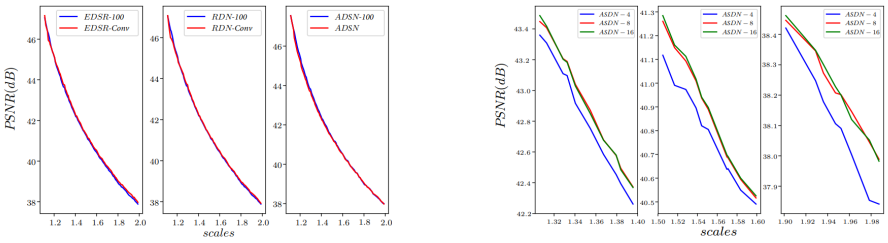
*Fixed-scale deep network* (**FSDN**) is a network similar to ASDN, with the only exception of a transposed convolution at the beginning of each IRB, which is fine tuned to a specific scale.

## 3.5 Results

### 3.5.1 Efficiency of the Laplacian Frequency Representation and density study.

Looking at [Figure 46] the Laplacian Frequency Representation (network trained using 11 IRBs: EDSR-Conv, RDN-Conv, ASDN) has the same performances of networks trained using pre-upsampling method (EDSR-100, RDN-100, ADSN-100 on 100 scales between 1 and 2).

We can see that the denser the pyramid the better but at some point it saturate and no benefit are achieved (ASDN-**x** where $x$ is the number of phases in the LFR so $L = x + 1$.)




(a) Study of Effectiveness

(b) Study of Density

Fig. 8: Study of Laplacian Frequency Representation

Figure 46: Effectiveness and density of LFR.

### 3.5.2 Efficiency of the recursive deployment and optimal N and r

The direct deployment uses networks trained with images upscaled directly to x2,x3,x4 instead recursive deployment uses networks trained twice using $x\sqrt{2}$, $x\sqrt{3}$,$x\sqrt{4}$.

From [Figure 47] we can see that the difference in performance is noticeable with lower scales and decrease with it and using the greatest scale at the beginning lead to better performance.

| Methods | Direct deployment | | | Recursive deployment | | |
|---|---|---|---|---|---|---|
| | ×2 | ×3 | ×4 | ×2 | ×3 | ×4 |
| VDSR-Conv | 37.57 | 33.77 | 31.56 | 36.86 | 33.70 | 31.50 |
| EDSR-Conv | 38.04 | 34.45 | 32.29 | 37.18 | 34.32 | 32.26 |
| RDN-Conv | 38.05 | 34.46 | 32.31 | 37.27 | 34.38 | 32.23 |
| Ours | 38.12 | 34.52 | 32.28 | 37.35 | 34.43 | 32.27 |

Table 3: PSNR of the recursive deployment and direct deployment on SR for ×2, ×3, ×4

(a) Study of the effectiveness of the recursive deployment.

| Scale(R) | Recursion(N) | UpscaleRatio(r) | PSNR |
|---|---|---|---|
| 3× | 2 | 1.732, 1.732 | 34.43 |
| | | 1.500, 2.000 | 34.19 |
| | | 2.000, 1.500 | **34.48** |
| | 3 | 1.442, 1.442, 1.442 | 33.18 |
| 4× | 2 | 2.000, 2.000 | **32.27** |
| | 3 | 1.587, 1.587, 1.587 | 31.96 |
| | | 1.800, 1.800, 1.234 | 32.16 |
| | | 2.000, 1.800, 1.100 | 32.24 |

(b) Optimal N and r for the recursive deployment.

Figure 47: Recursive deployment studies.

### 3.5.3 Ablation studies and performance versus parameters

Every component inside ADSN is important for achieving better performance as shown in Figure 48a.

The network is able to achieve state-of-the-art performance with a small amount of parameters (22M).

| Module | Different combination of CA, SA and SC | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| CA | × | × | × | √ | √ | √ | × | √ |
| SA | × | × | √ | × | √ | × | √ | √ |
| SC | × | √ | × | × | × | √ | √ | √ |
| PSNR | 37.92 | 37.96 | 37.93 | 37.95 | 37.97 | 37.99 | 37.97 | **38.01** |

(a) Ablation studies on channel attention (CA), spatial attention (SA) and global skip connection (SC).

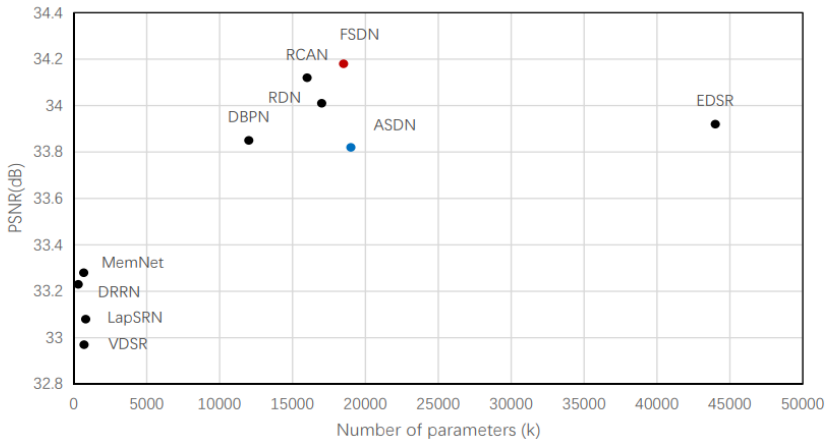



Fig. 9: Performance vs number of parameters. The results are evaluated with Set14 for 2× enlargement. Red indicates the best performance, and blue indicates the best performance among predefined upsampling methods

(b) PSNR versus number of parameters.

### 3.5.4 Quantitative and qualitative results

**Fixed scales** Overall ASDN performs better than any predefined upsampling method and FSDN has the best performance (this highlight the fact that having learnable upscaling method allow to achieve better performance).

Sometimes ASDN performs worse because some network are trained specifically for a particular scales with the particular dataset or it is worse than RCAN because has many channel attention modules that allow to improve the reconstruction.

Table 1: Quantitative evaluation of state-of-the-art SR algorithms. We report the average PSNR/SSIM for 2×, 3×, 4× and 8× SR. Red indicates the best performance, and blue indicates the best performance among predefined upsampling methods.

| scale | Algorithms | Set5 | | Set14 | | BSD100 | | Urban100 | | Manga109 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **PSNR** | **SSIM** | **PSNR** | **SSIM** | **PSNR** | **SSIM** | **PSNR** | **SSIM** | **PSNR** | **SSIM** |
| 2× | Bicubic | 33.64 | 0.929 | 30.22 | 0.868 | 29.55 | 0.842 | 26.66 | 0.841 | 30.84 | 0.935 |
| | SRCNN [4] | 36.65 | 0.954 | 32.29 | 0.903 | 31.36 | 0.888 | 29.52 | 0.895 | 35.72 | 0.968 |
| | VDSR [12] | 37.53 | 0.958 | 32.97 | 0.913 | 31.90 | 0.896 | 30.77 | 0.914 | 37.16 | 0.974 |
| | DRRN [7] | 37.74 | 0.959 | 33.23 | 0.914 | 32.05 | 0.897 | 31.23 | 0.919 | 37.52 | 0.976 |
| | LapSRN [13] | 37.52 | 0.959 | 33.08 | 0.913 | 31.80 | 0.895 | 30.41 | 0.910 | 37.27 | 0.974 |
| | MemNet [21] | 37.78 | 0.959 | 33.28 | 0.914 | 32.08 | 0.898 | 31.33 | 0.919 | 37.72 | 0.974 |
| | SRMDNF [27] | 37.79 | 0.960 | 33.32 | 0.916 | 32.05 | 0.899 | 31.33 | 0.920 | 38.07 | 0.976 |
| | ASDN(ours) | 38.12 | 0.961 | 33.82 | 0.919 | 32.30 | 0.901 | 32.47 | 0.931 | 39.16 | 0.978 |
| | EDSR [15] | 38.11 | 0.960 | 33.92 | 0.920 | 32.32 | 0.901 | 32.93 | 0.935 | 39.10 | 0.976 |
| | RDN [29] | 38.24 | 0.961 | 34.01 | 0.921 | 32.34 | 0.902 | 32.96 | 0.936 | 39.19 | 0.978 |
| | DBPN [6] | 38.09 | 0.961 | 33.85 | 0.920 | 32.27 | 0.900 | 32.55 | 0.932 | 38.89 | 0.978 |
| | RCAN [28] | 38.27 | 0.961 | 34.12 | 0.922 | 32.41 | 0.903 | 33.34 | 0.938 | 39.44 | 0.979 |
| | FSDN(ours) | 38.27 | 0.961 | 34.18 | 0.923 | 32.41 | 0.903 | 33.13 | 0.937 | 39.49 | 0.979 |
| 3× | Bicubic | 30.39 | 0.867 | 27.53 | 0.774 | 27.20 | 0.738 | 24.47 | 0.737 | 26.99 | 0.859 |
| | SRCNN [4] | 32.75 | 0.909 | 29.30 | 0.822 | 28.41 | 0.786 | 26.25 | 0.801 | 30.59 | 0.914 |
| | VDSR [12] | 33.66 | 0.921 | 29.77 | 0.831 | 28.82 | 0.798 | 27.41 | 0.830 | 32.01 | 0.934 |
| | DRRN [7] | 34.03 | 0.924 | 29.96 | 0.835 | 28.95 | 0.800 | 27.53 | 0.764 | 32.42 | 0.939 |
| | LapSRN [13] | 33.82 | 0.922 | 29.87 | 0.832 | 28.82 | 0.798 | 27.07 | 0.828 | 32.21 | 0.935 |
| | MemNet [21] | 34.09 | 0.925 | 30.00 | 0.835 | 28.96 | 0.800 | 27.57 | 0.839 | 32.51 | 0.937 |
| | SRMDNF [27] | 34.12 | 0.925 | 30.04 | 0.838 | 28.97 | 0.802 | 27.57 | 0.839 | 33.00 | 0.940 |
| | ASDN(ours) | 34.48 | 0.928 | 30.35 | 0.843 | 29.18 | 0.808 | 28.45 | 0.858 | 33.87 | 0.947 |
| | EDSR [15] | 34.65 | 0.928 | 30.52 | 0.846 | 29.25 | 0.809 | 28.80 | 0.865 | 34.17 | 0.948 |
| | RDN [29] | 34.71 | 0.929 | 30.57 | 0.847 | 29.26 | 0.809 | 28.80 | 0.865 | 34.13 | 0.948 |
| | RCAN [28] | 34.74 | 0.930 | 30.65 | 0.848 | 29.32 | 0.811 | 29.09 | 0.870 | 34.44 | 0.949 |
| | FSDN(ours) | 34.75 | 0.930 | 30.63 | 0.848 | 29.33 | 0.811 | 28.98 | 0.868 | 34.53 | 0.950 |
| 4× | Bicubic | 28.42 | 0.810 | 26.10 | 0.704 | 25.96 | 0.669 | 23.15 | 0.660 | 24.92 | 0.789 |
| | SRCNN [4] | 30.49 | 0.862 | 27.61 | 0.754 | 26.91 | 0.712 | 24.53 | 0.724 | 27.66 | 0.858 |
| | VDSR [12] | 31.35 | 0.882 | 28.03 | 0.770 | 27.32 | 0.730 | 25.18 | 0.750 | 28.82 | 0.886 |
| | DRRN [7] | 31.68 | 0.889 | 28.21 | 0.772 | 27.38 | 0.728 | 25.44 | 0.764 | 29.18 | 0.891 |
| | MemNet [21] | 31.74 | 0.889 | 28.26 | 0.772 | 27.40 | 0.728 | 25.50 | 0.763 | 29.42 | 0.894 |
| | SRMDNF [27] | 31.96 | 0.892 | 28.35 | 0.778 | 27.49 | 0.734 | 25.68 | 0.773 | 30.09 | 0.902 |
| | ASDN(ours) | 32.27 | 0.896 | 28.66 | 0.784 | 27.65 | 0.740 | 26.27 | 0.792 | 30.91 | 0.913 |
| | LapSRN [13] | 31.54 | 0.885 | 28.19 | 0.772 | 27.32 | 0.727 | 25.21 | 0.756 | 29.46 | 0.890 |
| | EDSR [15] | 32.46 | 0.896 | 28.80 | 0.788 | 27.71 | 0.742 | 26.64 | 0.803 | 31.02 | 0.915 |
| | RDN [29] | 32.47 | 0.899 | 28.81 | 0.787 | 27.72 | 0.742 | 26.61 | 0.803 | 31.00 | 0.915 |
| | DBPN [6] | 32.42 | 0.898 | 28.76 | 0.786 | 27.68 | 0.740 | 26.38 | 0.796 | 30.91 | 0.914 |
| | RCAN [28] | 32.63 | 0.900 | 28.87 | 0.789 | 27.77 | 0.744 | 26.82 | 0.809 | 31.22 | 0.917 |
| | FSDN(ours) | 32.63 | 0.900 | 28.89 | 0.789 | 27.79 | 0.744 | 26.79 | 0.807 | 31.44 | 0.919 |
| 8× | Bicubic | 24.40 | 0.658 | 23.10 | 0.566 | 23.97 | 0.548 | 20.74 | 0.516 | 21.47 | 0.650 |
| | SRCNN [4] | 25.33 | 0.690 | 23.76 | 0.591 | 24.13 | 0.566 | 21.29 | 0.544 | 22.46 | 0.695 |
| | VDSR [12] | 25.93 | 0.724 | 24.26 | 0.614 | 24.49 | 0.583 | 21.70 | 0.571 | 23.16 | 0.725 |
| | LapSRN [13] | 26.15 | 0.738 | 24.35 | 0.620 | 24.54 | 0.586 | 21.81 | 0.581 | 23.39 | 0.735 |
| | MemNet [21] | 26.16 | 0.741 | 24.38 | 0.619 | 24.58 | 0.584 | 21.89 | 0.583 | 23.56 | 0.738 |
| | ASDN(ours) | 27.02 | 0.776 | 24.99 | 0.641 | 24.82 | 0.600 | 22.57 | 0.620 | 24.73 | 0.748 |
| | EDSR [15] | 26.96 | 0.776 | 24.91 | 0.642 | 24.81 | 0.599 | 22.51 | 0.622 | 24.69 | 0.784 |
| | DBPN [6] | 27.21 | 0.784 | 25.13 | 0.648 | 24.88 | 0.601 | 22.73 | 0.631 | 25.14 | 0.799 |
| | RCAN [28] | 27.31 | 0.788 | 25.23 | 0.651 | 24.98 | 0.606 | 23.00 | 0.645 | 25.24 | 0.803 |
| | FSDN(ours) | 27.33 | 0.789 | 25.24 | 0.651 | 24.98 | 0.604 | 22.90 | 0.638 | 25.24 | 0.803 |

Figure 49: Comparison of quantitative results using fixed scales.

**Any-scale** ASDN performs better than state-of-the-art for upsampling images using decimal scales used or not for training.

Table 2: Results of any-scale SR on different methods tested on BSD100. The first row shows the results of Laplacian pyramid levels and the second row demonstrates SR performance on randomly selected scales and boundary condition. Red indicates the best performance, and blue indicates the second best performance.

| Method \ Scale | X1.1 | X1.2 | X1.3 | X1.4 | X1.5 | X1.6 | X1.7 | X1.8 | X1.9 |
|---|---|---|---|---|---|---|---|---|---|
| Bicubic | 36.56 | 35.01 | 33.84 | 32.93 | 32.14 | 31.49 | 30.90 | 30.38 | 29.97 |
| VDSR-Conv | 42.13 | 39.52 | 37.88 | 36.53 | 35.42 | 34.50 | 33.72 | 33.03 | 32.41 |
| EDSR-Conv | 42.92 | 40.11 | 38.33 | 36.93 | 35.79 | 34.85 | 34.06 | 33.38 | 32.75 |
| RDN-Conv | 42.86 | 40.04 | 38.25 | 36.86 | 35.72 | 34.78 | 33.99 | 33.29 | 32.67 |
| Meta-EDSR [8] | 42.72 | 39.92 | 38.16 | 36.84 | 35.78 | 34.83 | 34.06 | 33.36 | 32.78 |
| Meta-RDN [8] | 42.82 | 40.40 | 38.28 | 36.95 | 35.86 | 34.90 | 34.13 | 33.45 | 32.86 |
| ASDN(ours) | 43.05 | 40.24 | 38.42 | 37.02 | 35.87 | 34.92 | 34.14 | 33.46 | 32.86 |

| Method \ Scale | X2.0 | X2.8 | X4.0 | X5.7 | X8.0 | X11.3 | X16.0 | X22.6 | X32.0 |
|---|---|---|---|---|---|---|---|---|---|
| Bicubic | 29.55 | 27.53 | 25.96 | 24.96 | 23.67 | 22.65 | 21.73 | 20.73 | 19.90 |
| VDSR-Conv | 31.89 | 29.23 | 27.25 | 25.56 | 24.58 | 23.49 | 22.47 | 21.39 | 20.38 |
| EDSR-Conv | 32.23 | 29.54 | 27.58 | 26.01 | 24.78 | 23.65 | 22.63 | 21.55 | 20.53 |
| RDN-Conv | 32.07 | 29.47 | 27.51 | 25.94 | 24.72 | 23.60 | 22.58 | 21.50 | 20.51 |
| Meta-EDSR [8] | 32.26 | 29.61 | 27.67 | - | - | - | - | - | - |
| Meta-RDN [8] | 32.35 | 29.67 | 27.75 | - | - | - | - | - | - |
| ASDN(ours) | 32.30 | 29.63 | 27.65 | 26.07 | 24.85 | 23.70 | 22.66 | 21.59 | 20.55 |

(a) Comparison of quantitative results using decimal scales.

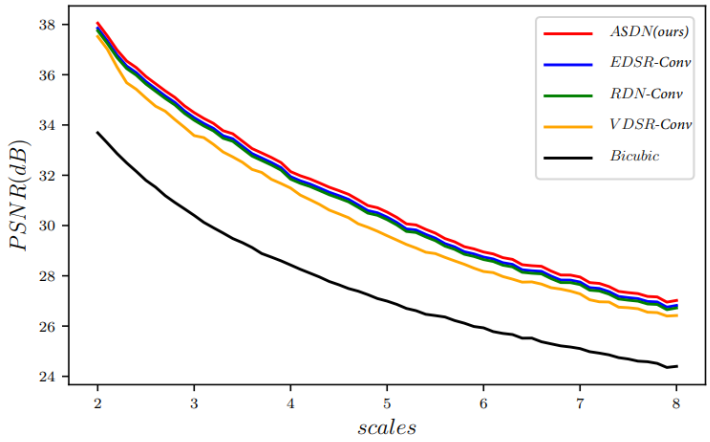



Fig. 4: PSNR comparison of ASDN with other works within the continuous scale range $(\times 2, \times 8]$ on Set5

(b) Comparison of quantitative results using decimal scales between 2 and 8.

**Qualitative results** The visual quality of ASDN is similar to state-of-the-art and FSDN outperforms state-of-the-art networks.

Fig. 5: Qualitative comparisons of our models with other works on ×2 super-resolution. Red indicates the best performance, and blue indicates the second best

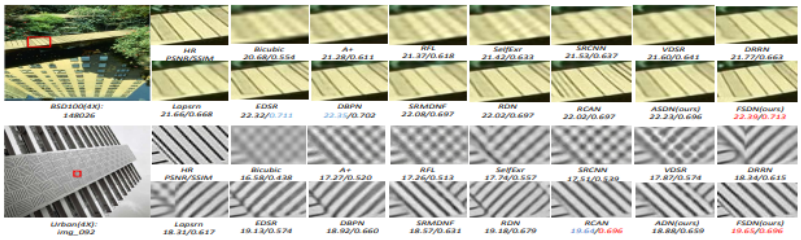


Fig. 6: Qualitative comparisons of our models with other works on ×4 super-resolution. Red indicates the best performance, and blue indicates the second best

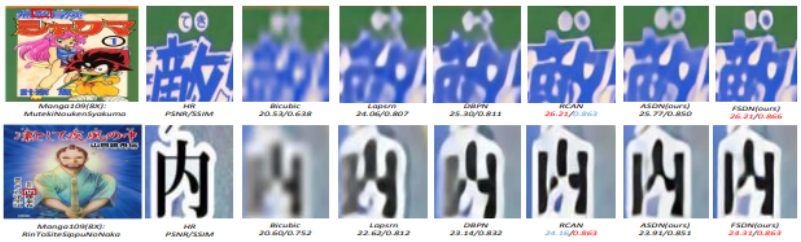
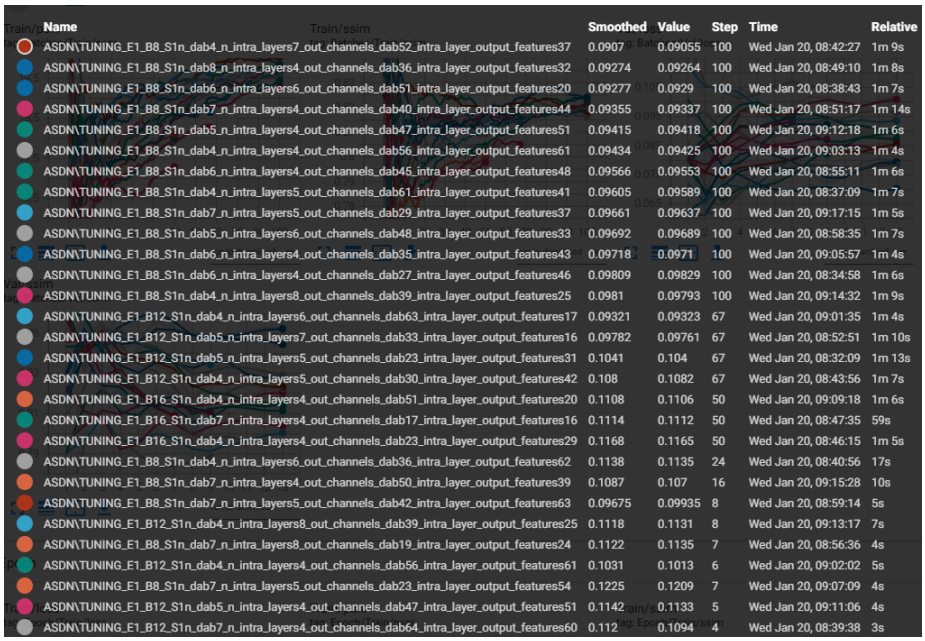


Fig. 7: Qualitative comparisons of our models with other works on ×8 super-resolution. Red indicates the best performance, and blue indicates the second best

Figure 51: Qualitative results of ASDN and FSDN on different datasets.



| Name | Smoothed | Value | Step | Time | Relative |
|---|---|---|---|---|---|
| ASDN\TUNING_E1_B8_S1n_dab4_n_intra_layers7_out_channels_dab52_intra_layer_output_features37 | 0.0907 | 0.09055 | 100 | Wed Jan 20, 08:42:27 | 1m 9s |
| ASDN\TUNING_E1_B8_S1n_dab8_n_intra_layers4_out_channels_dab36_intra_layer_output_features32 | 0.09274 | 0.09264 | 100 | Wed Jan 20, 08:49:10 | 1m 8s |
| ASDN\TUNING_E1_B8_S1n_dab6_n_intra_layers6_out_channels_dab51_intra_layer_output_features20 | 0.09277 | 0.0929 | 100 | Wed Jan 20, 08:38:43 | 1m 7s |
| ASDN\TUNING_E1_B8_S1n_dab7_n_intra_layers4_out_channels_dab40_intra_layer_output_features44 | 0.09355 | 0.09337 | 100 | Wed Jan 20, 08:51:17 | 1m 14s |
| ASDN\TUNING_E1_B8_S1n_dab5_n_intra_layers4_out_channels_dab47_intra_layer_output_features51 | 0.09415 | 0.09418 | 100 | Wed Jan 20, 09:12:18 | 1m 6s |
| ASDN\TUNING_E1_B8_S1n_dab4_n_intra_layers4_out_channels_dab56_intra_layer_output_features61 | 0.09434 | 0.09425 | 100 | Wed Jan 20, 09:03:13 | 1m 4s |
| ASDN\TUNING_E1_B8_S1n_dab6_n_intra_layers4_out_channels_dab45_intra_layer_output_features48 | 0.09566 | 0.09553 | 100 | Wed Jan 20, 08:55:11 | 1m 6s |
| ASDN\TUNING_E1_B8_S1n_dab4_n_intra_layers5_out_channels_dab61_intra_layer_output_features41 | 0.09605 | 0.09589 | 100 | Wed Jan 20, 08:37:09 | 1m 7s |
| ASDN\TUNING_E1_B8_S1n_dab7_n_intra_layers5_out_channels_dab29_intra_layer_output_features37 | 0.09661 | 0.09637 | 100 | Wed Jan 20, 09:17:15 | 1m 5s |
| ASDN\TUNING_E1_B8_S1n_dab5_n_intra_layers6_out_channels_dab48_intra_layer_output_features33 | 0.09692 | 0.09689 | 100 | Wed Jan 20, 08:58:35 | 1m 7s |
| ASDN\TUNING_E1_B8_S1n_dab6_n_intra_layers4_out_channels_dab35_intra_layer_output_features43 | 0.09718 | 0.0971 | 100 | Wed Jan 20, 09:05:57 | 1m 4s |
| ASDN\TUNING_E1_B8_S1n_dab6_n_intra_layers4_out_channels_dab27_intra_layer_output_features46 | 0.09809 | 0.09829 | 100 | Wed Jan 20, 08:34:58 | 1m 6s |
| ASDN\TUNING_E1_B8_S1n_dab4_n_intra_layers8_out_channels_dab39_intra_layer_output_features25 | 0.0981 | 0.09793 | 100 | Wed Jan 20, 09:14:32 | 1m 9s |
| ASDN\TUNING_E1_B12_S1n_dab4_n_intra_layers6_out_channels_dab63_intra_layer_output_features17 | 0.09321 | 0.09323 | 67 | Wed Jan 20, 09:01:35 | 1m 4s |
| ASDN\TUNING_E1_B12_S1n_dab5_n_intra_layers7_out_channels_dab33_intra_layer_output_features16 | 0.09782 | 0.09761 | 67 | Wed Jan 20, 08:52:51 | 1m 10s |
| ASDN\TUNING_E1_B12_S1n_dab5_n_intra_layers5_out_channels_dab23_intra_layer_output_features31 | 0.1041 | 0.104 | 67 | Wed Jan 20, 08:32:09 | 1m 13s |
| ASDN\TUNING_E1_B12_S1n_dab4_n_intra_layers5_out_channels_dab30_intra_layer_output_features42 | 0.108 | 0.1082 | 67 | Wed Jan 20, 08:43:56 | 1m 7s |
| ASDN\TUNING_E1_B16_S1n_dab4_n_intra_layers4_out_channels_dab51_intra_layer_output_features20 | 0.1108 | 0.1106 | 50 | Wed Jan 20, 09:09:18 | 1m 6s |
| ASDN\TUNING_E1_B16_S1n_dab7_n_intra_layers4_out_channels_dab17_intra_layer_output_features16 | 0.1114 | 0.1112 | 50 | Wed Jan 20, 08:47:35 | 59s |
| ASDN\TUNING_E1_B16_S1n_dab4_n_intra_layers4_out_channels_dab23_intra_layer_output_features29 | 0.1168 | 0.1165 | 50 | Wed Jan 20, 08:46:15 | 1m 5s |
| ASDN\TUNING_E1_B8_S1n_dab4_n_intra_layers6_out_channels_dab36_intra_layer_output_features62 | 0.1138 | 0.1135 | 24 | Wed Jan 20, 08:40:56 | 17s |
| ASDN\TUNING_E1_B8_S1n_dab7_n_intra_layers4_out_channels_dab50_intra_layer_output_features39 | 0.1087 | 0.107 | 16 | Wed Jan 20, 09:15:28 | 10s |
| ASDN\TUNING_E1_B8_S1n_dab7_n_intra_layers5_out_channels_dab42_intra_layer_output_features63 | 0.09675 | 0.09935 | 8 | Wed Jan 20, 08:59:14 | 5s |
| ASDN\TUNING_E1_B12_S1n_dab4_n_intra_layers8_out_channels_dab39_intra_layer_output_features25 | 0.1118 | 0.1131 | 8 | Wed Jan 20, 09:13:17 | 7s |
| ASDN\TUNING_E1_B8_S1n_dab7_n_intra_layers8_out_channels_dab19_intra_layer_output_features24 | 0.1122 | 0.1135 | 7 | Wed Jan 20, 08:56:36 | 4s |
| ASDN\TUNING_E1_B12_S1n_dab4_n_intra_layers4_out_channels_dab56_intra_layer_output_features61 | 0.1031 | 0.1013 | 6 | Wed Jan 20, 09:02:02 | 5s |
| ASDN\TUNING_E1_B8_S1n_dab7_n_intra_layers5_out_channels_dab23_intra_layer_output_features54 | 0.1225 | 0.1209 | 7 | Wed Jan 20, 09:07:09 | 4s |
| ASDN\TUNING_E1_B12_S1n_dab5_n_intra_layers4_out_channels_dab47_intra_layer_output_features51 | 0.1142 | 0.1133 | 5 | Wed Jan 20, 09:11:06 | 4s |
| ASDN\TUNING_E1_B12_S1n_dab7_n_intra_layers4_out_channels_dab64_intra_layer_output_features60 | 0.112 | 0.1094 | 4 | Wed Jan 20, 08:39:38 | 3s |

(a) Training time of different random configurations. There are many other configurations missing due to *out of memory* error.

| Name | Smoothed | Value | Step | Time | Relative |
|---|---|---|---|---|---|
| ASDN\BATCH_TIME_E1_B4_S2 | 0.08426 | 0.08408 | 200 | Wed Jan 20, 10:28:47 | 21m 10s |
| ASDN\BATCH_TIME_E1_B8_S1n_dab5_n_intra_layers5_out_channels_dab32_intra_layer_output_features40 | 0.09596 | 0.0957 | 100 | Wed Jan 20, 11:22:07 | 1m 6s |
| ASDN\BATCH_TIME_E1_B8_S2n_dab5_n_intra_layers5_out_channels_dab32_intra_layer_output_features40 | 0.1 | 0.1003 | 100 | Wed Jan 20, 11:23:44 | 1m 24s |

(b) Comparison between the training time of 1 epoch of the default configuration, using the trick and not using the trick.

Figure 52: Trining time of 1 epoch. The name contains: E is *epoch*, B is *batch size*, S is *save checkpoint* and the network parameters.

# 4 Project

## 4.1 Hardware

The following hardware was used:

- GTX1060 (6 GB)

## 4.2 Architecture configuration

**ASDN** is using a bi-dense connection this means a lot of memory is used for storing intermediate activations as well as gradients: the default configuration, number of dense attention block 16 and number of intra connected layers 8, is not suitable for my computer.

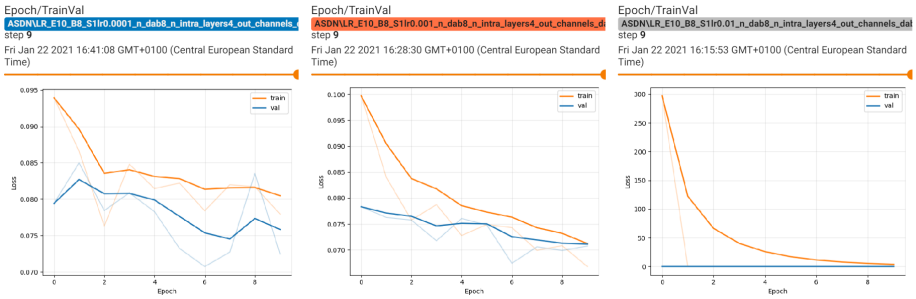
In order to train the default configuration an implementation trick can be used (**save checkpoint**[1]): avoid to store gradient and activations during the forward pass and recompute them during the backward pass; but this increase the training time of a **single batch** as we can see from Figure 52b

Among many configurations tried [Figure 52a] the final configuration chosen uses half the parameters with respect to the default configuration:

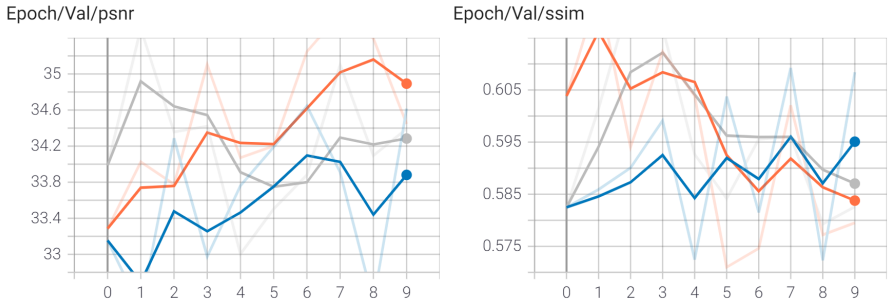
| | Project architecture | Paper architecture |
|---|---|---|
| DABs | 8 | 16 |
| IDBs | 4 | 8 |
| Features DAB | 32 | 64 |
| Features IDB | 32 | 64 |

## 4.3 Experiments

### 4.3.1 The initial LR




(a) Loss values with lr: 0.01 (right), 0.001(middle), 0.0001(left)




(b) PSNR and SSIM values with lr: 0.01 (gray), 0.001 (orange), 0.0001 (blue)

Figure 53: Experiments with different *lr*.

The initial learning rate is set to **0.001** based on the experiments done (Figure 53).

[1] PyTorch checkpoint

| | Paper model | Base model(dark blue) | Complex model (light blue) |
|---|---|---|---|
| Total params | 22,405,168 | 828,928 | 1,111,270 |
| Trainable params | 22,405,168 | 828,928 | 1,111,270 |
| Non trainable params | 0 | 0 | 0 |
| Input size (MB) | 0.84 | 0.84 | 0.84 |
| Forward/Backward pass size (MB) | 10231.38 | 4911.24 | 5281.37 |
| Params size (MB) | 85.47 | 3.16 | 4.24 |
| Estimated total size | 10317.70 | 4915.24 | 5286.45 |

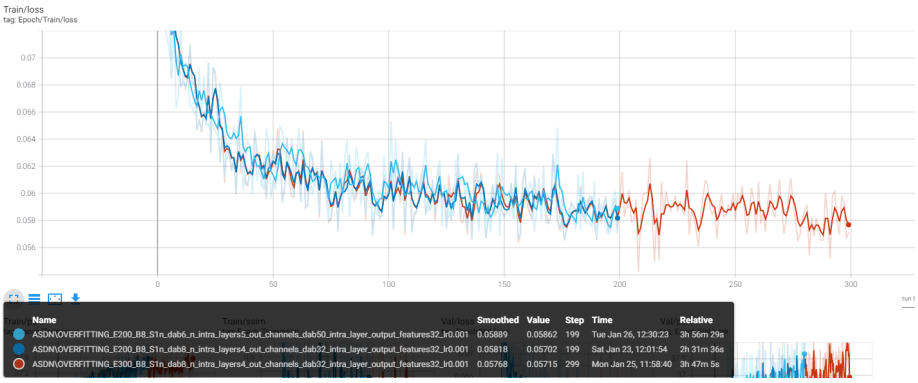
Table 1: Comparison complexity between models.

#### 4.3.2 Overfitting

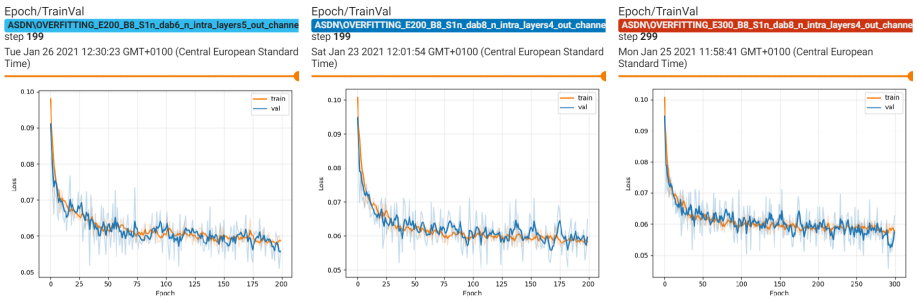
In order to see if the network was capable to learn the mapping some overfitting experiments were done [Figure 54]: neither increasing the epochs (red) nor the complexity of the model(light blue) lead to overfitting with respect to the baseline (dark blue) [Table 1].

Moreover the noisy loss could be caused by the regularization action of the small batch size and by training data downsampled using a bilinear interpolation which can be seen as an data augmentation used for generalization and regularization of the network.

An lr scheduler was used in order to try to decrease further the loss but without success (Figure 55).

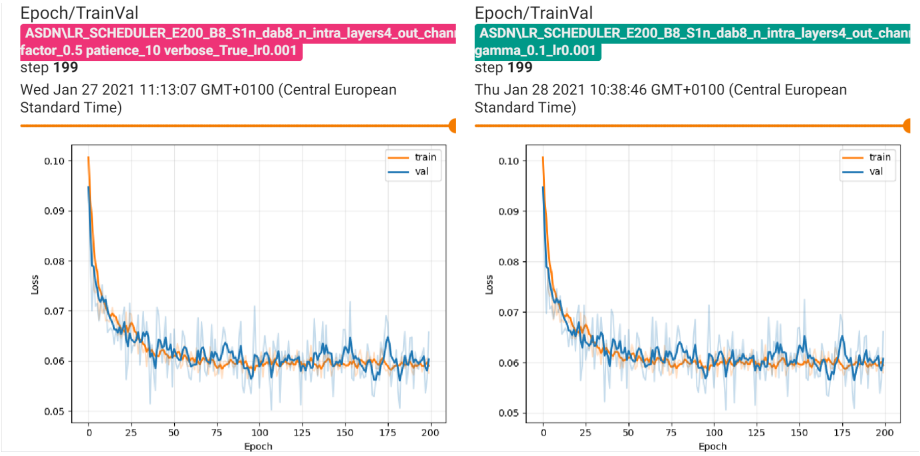



(a) Loss.

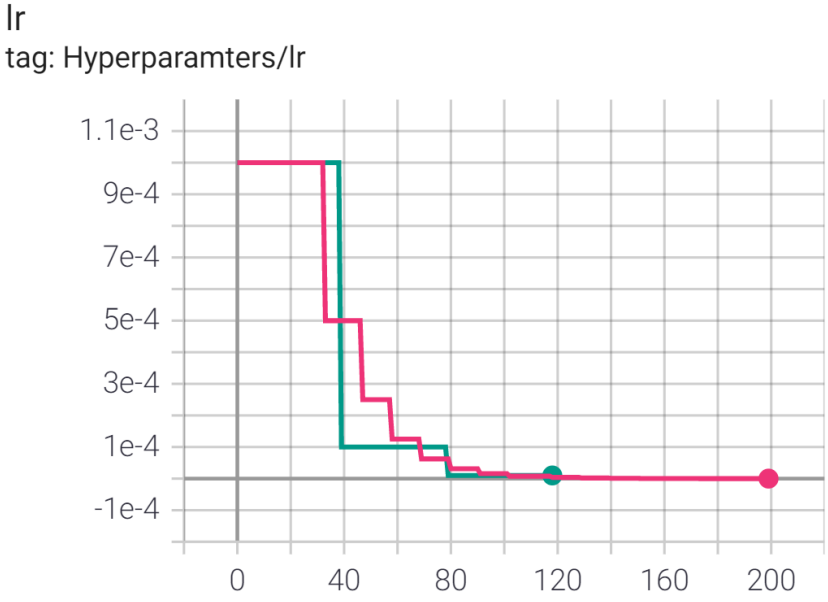



(b) Train-Val loss.

Figure 54: Overfitting experiments.




(a) Train-Val loss with StepLR(green) and ReduceLROnPlateau(pink).




(b) LR trend with StepLR(green) and ReduceLROnPlateau(pink).

Figure 55: LR scheduler experiments.

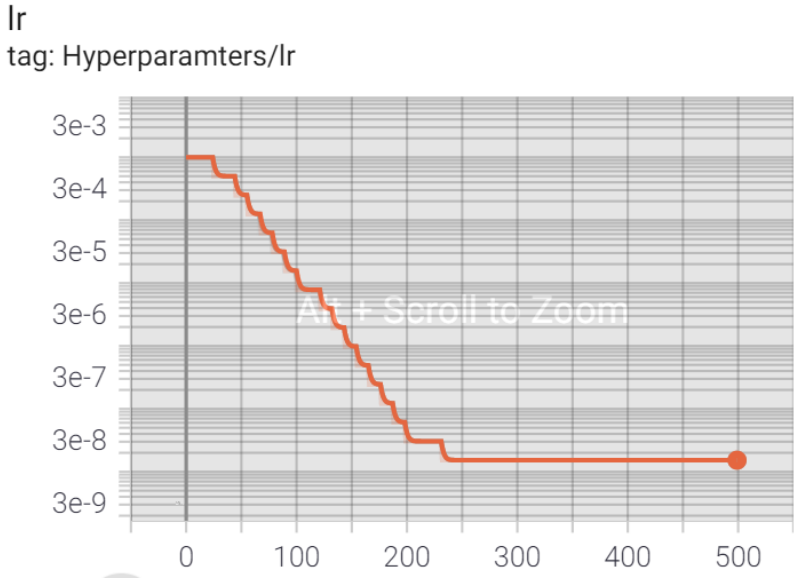
### 4.3.3 Training

The network was trained for 500 epochs without and with data augmentation (random vertical flip, random horizontal flip, random 90 degree rotation) in order to increase patches seen without increasing the number of images and therefore without increasing training time and aggressively reducing the lr in

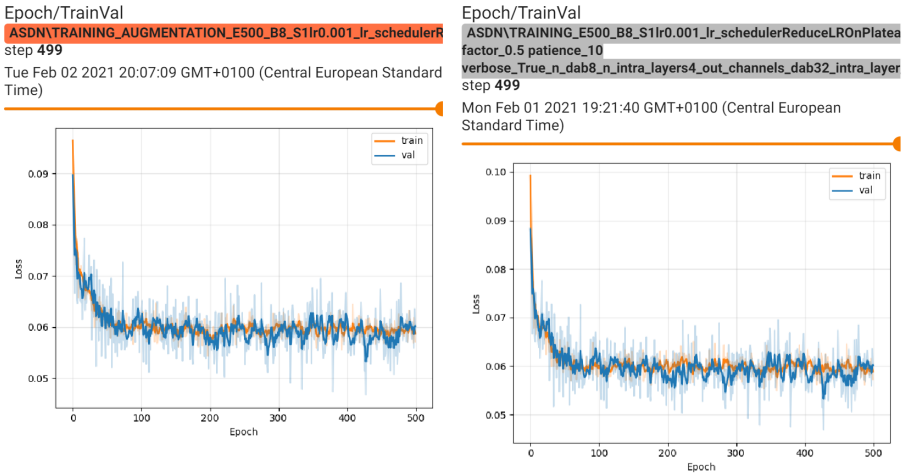
order to see an improvement of the loss.

In both case the loss is steady to ~0.6 and no real difference between outputs can be seen [Figure 57] .


(a) Training time.




(b) Learning rate over time.




(c) Train-Val loss.

Figure 56: Training with 500 epochs.


(a) Input.


(b) Output.

Figure 57: Validation images at 500-th epoch.

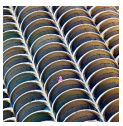
#### 4.3.4 Testing

The following are the results of the base model during testing with:

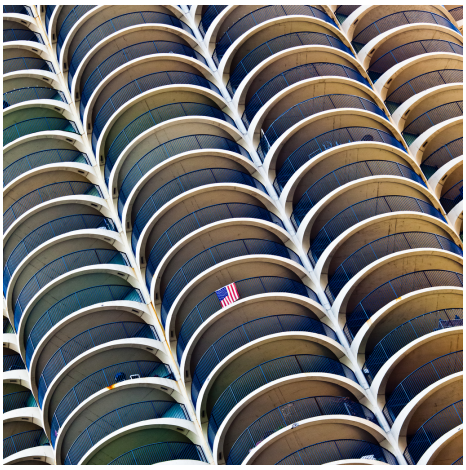
| Dataset | PSNR | SSIM |
|---|---|---|
| BSDS100 | 13.489 | 0.900 |
| Manga109 | 11.834 | 0.890 |
| Set14 | 13.220 | 0.905 |
| Set5 | 11.893 | 0.875 |
| Urban100 | 12.665 | 0.985 |
| historical[2] | 13.821 | 0.922 |

The only comparison is done with bicubic interpolation. The network is able to reconstruct sharp edges but unable to restore all the details from the original image.

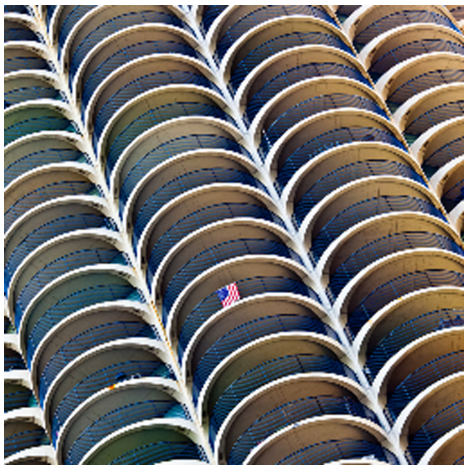
[1]Ground truth images were already LR images



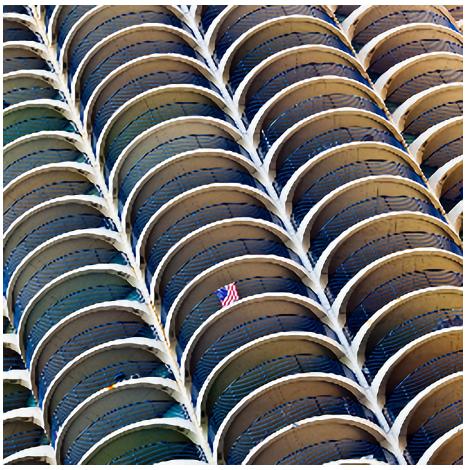
(a) Input



(b) Ground truth.



(c) Bicubic.



(d) Output.

Figure 58: Base model applied to Urban100.

(a) Input


(b) Ground truth.


(c) Bicubic.


(d) Output.

Figure 59: Base model applied to Manga109.

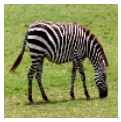
(a) Input

(b) Ground truth.

(c) Bicubic.

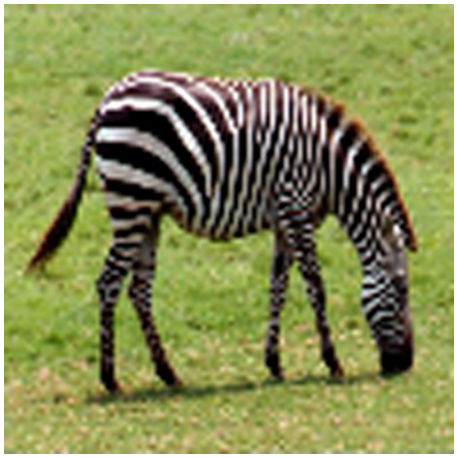
(d) Output.

Figure 60: Base model applied to Set5.



(a) Input

(b) Ground truth.



(c) Bicubic.

(d) Output.

Figure 61: Base model applied to Set14.

(a) Input

(b) Ground truth.

(c) Bicubic.

(d) Output.

Figure 62: Base model applied to BSDS100.

(a) Input

(b) Ground truth.

(c) Bicubic.

(d) Output.

Figure 63: Base model applied to historical.

# References


[1] J. Shen, Y. Wang, and J. Zhang, "Asdn: A deep convolutional network for arbitrary scale image super-resolution," *ArXiv*, vol. abs/2010.02414, 2020.

[2] Y. Wang, J. Shen, and J. Zhang, "Deep bi-dense networks for image super-resolution," *CoRR*, vol. abs/1810.04873, 2018.

[3] Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong, and Y. Fu, "Image super-resolution using very deep residual channel attention networks," *CoRR*, vol. abs/1807.02758, 2018.

[4] Y. Hu, J. Li, Y. Huang, and X. Gao, "Channel-wise and spatial feature modulation network for single image super-resolution," *CoRR*, vol. abs/1809.11130, 2018.

[5] W. Lai, J. Huang, N. Ahuja, and M. Yang, "Deep laplacian pyramid networks for fast and accurate super-resolution," *CoRR*, vol. abs/1704.03915, 2017.

[6] W. Lai, J. Huang, N. Ahuja, and M. Yang, "Fast and accurate image super-resolution with deep laplacian pyramid networks," *CoRR*, vol. abs/1710.01992, 2017.

[7] X. Hu, H. Mu, X. Zhang, Z. Wang, T. Tan, and J. Sun, "Meta-sr: A magnification-arbitrary network for super-resolution," *CoRR*, vol. abs/1903.00875, 2019.

[8] P. Burt and E. Adelson, "The laplacian pyramid as a compact image code," *IEEE Transactions on Communications*, vol. 31, no. 4, pp. 532–540, 1983.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," *CoRR*, vol. abs/1603.05027, 2016.

[10] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network," *CoRR*, vol. abs/1609.05158, 2016.

[11] Hong Chang, Dit-Yan Yeung, and Yimin Xiong, "Super-resolution through neighbor embedding," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 1, pp. I–I, 2004.

[12] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *CoRR*, vol. abs/1501.00092, 2015.

[13] Y. Zhang, Y. Tian, Y. Kong, B. Zhong, and Y. Fu, "Residual dense network for image restoration," *CoRR*, vol. abs/1812.10477, 2018.

[14] J. Kim, J. K. Lee, and K. M. Lee, "Deeply-recursive convolutional network for image super-resolution," *CoRR*, vol. abs/1511.04491, 2015.

[15] Y. Tai, J. Yang, and X. Liu, "Image super-resolution via deep recursive residual network," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2790–2798, 2017.

[16] Q. Fan, D. Chen, L. Yuan, G. Hua, N. Yu, and B. Chen, "Decouple learning for parameterized image operators," *CoRR*, vol. abs/1807.08186, 2018.

[17] W. Shi, J. Caballero, L. Theis, F. Huszar, A. P. Aitken, C. Ledig, and Z. Wang, "Is the deconvolution layer the same as a convolutional layer?," *CoRR*, vol. abs/1609.07009, 2016.