Machine Learning for Computer Vision - Paper presentation & Project work

Pierpasquale Colagrande
Alma Mater Studiorum - University of Bologna
pierpasqu.colagrande@studio.unibo.it

Abstract

The following project is a Python implementation of the paper "MPRNet: Multi-Path Residual Network for Lightweight Image Super Resolution"[1] published by Mehri et al. in November, 2020. In addition to that, the project consists in training and testing the model with a series of different Super Resolution benchmark datasets, including datasets that were not tested in the paper. These additional datasets are Manga109 [2, 3], SunHays80 [4] and historical dataset. This report consists in an introduction about the Super Resolution task, with research background explanation of related work. Following, the paper is explained and details about the implementation and training/testing phases are given. Finally, the qualitative and quantitative results of the various tests are disclosed.

1 Introduction

Super Resolution is the task of computing high resolution equivalents of images with a low resolution. More specifically, super resolution networks receive as input a low resolution image and an scaling factor and output the same image in high resolution, upscaled by the specified scale. Single image super resolution aims to restore a high-resolution image from its degraded low-resolution version.

1.1 Naive upscaling

Upscaling can also be achieved via interpolation algorithms such as bicubic, bilinear, nearest neighbor etc. However these techniques tend to produce upscaled images that don't have a quality that is good enough to consider the output image a high resolution image. For example, upscaling through bicubic interpolation produces images that appear blurry and, depending on the scaling factor, this effect can be more or less evident. Therefore, bicubic tend to work better with small upscaling factors (e.g. a 2x upscaling), because the image will look better, however when it comes to images that require a higher scaling factor (e.g. 4x or higher upscalings) the output images appear very blurry and very low quality in general. This effect can also be more or less tolerable depending on the context of the image: for example, images that have a lot of details and intricate textures, such as urban environments, may not be suitable for bicubic upscaling because the quality of the produced image will not be good enough. Same happens for images with text, output images may be not readable.

1.2 Deep Learning based upscaling

In this setting, a solution to improving the quality of the images comes with the use of Deep Learning: by means of it, it is possible to teach Neural Networks to produce an upscaled image with a higher quality with respect to their non-learning produced counterparts. Deep learning can be used to learn representations of textures and to identify the context of the images in order to produce images with a better quality by retrieving missing scene details.

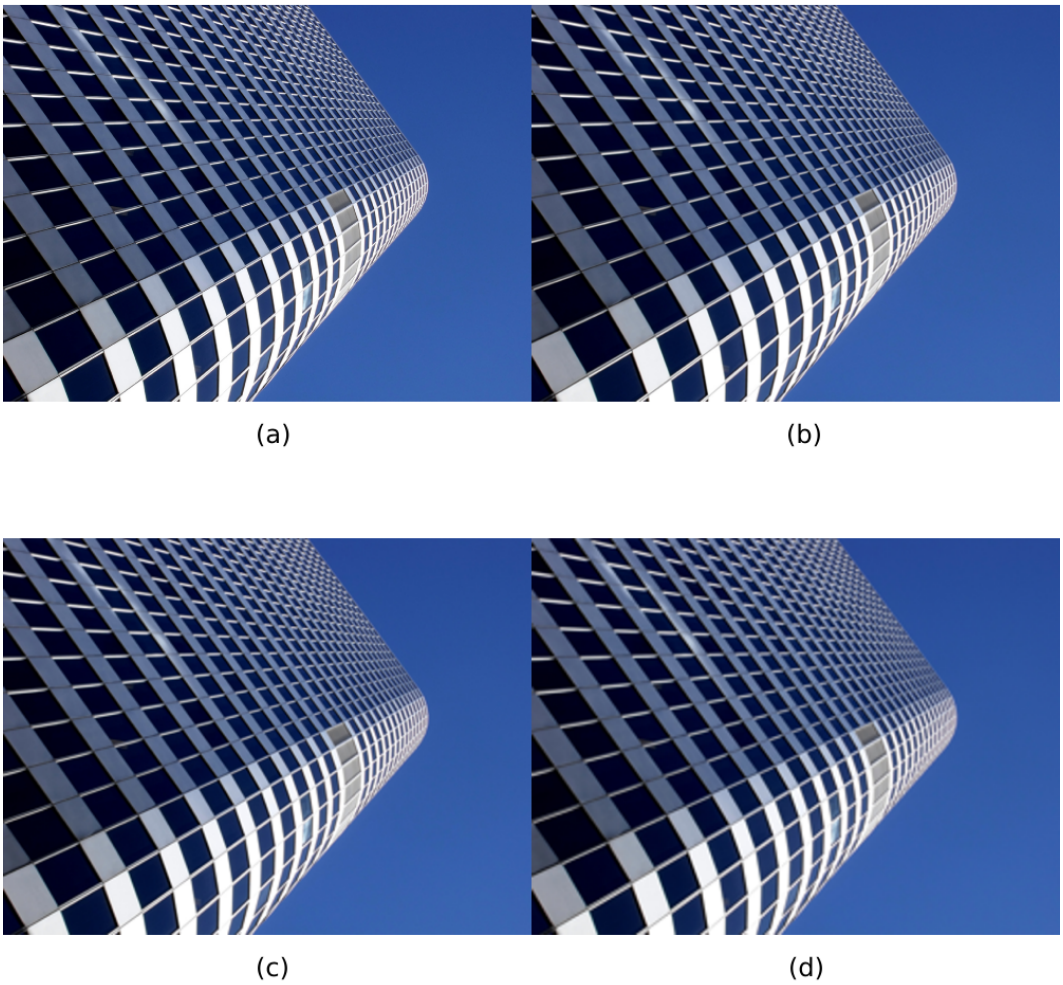


Figure 1: Comparison between the original high resolution image (a) and the images upscaled by 2x (b), 3x (c) and 4x (d) using bicubic interpolation. All images have the same resolution. Image from Urban100 [5] dataset.

2 Related work

Single image super resolution is an active research field and its applications range from HD TVs to smartphones and portable devices. It also leads to improvements in various tasks as face recognition, object detection etc. SR problem is challenging because as the upscaling factor increases, the complexity of the problem does it too. In recent years, CNNs have been used to solve the super resolution task by designing new and deeper architectures. These very deep networks allow to achieve SOTA results and higher PSNR scores but the depth and complexity of these networks increase memory usage and computational cost for the sake of achieving outstanding SOTA performances. Lightweight SR networks are thus important for real-world application of SR.

2.1 Deep Learning based Super Resolution

One of the first works using CNN to tackle the SR task was presented by Dong et al. [6]. The network introduced by Dong et al. was called SRCNN. The network receives a bicubic upscaled grayscale (Y channel of YCbCr color space) LR image and uses 3 layers to implement a sparse-coding-based SR method:

1. patch & feature extraction: the first layer extracts patches from the LR image and computes a set of feature maps for each patch to represent them
2. non-linear mapping: the second layer is a convolution that maps these low resolution feature maps to high resolution feature maps non-linearly (using ReLU)
3. reconstruction: the third layer combines these predicted feature maps to produce the output image

The problem with this network is that it receives an already upscaled image, so it works at the output resolution since the beginning and this requires an extra computational cost.

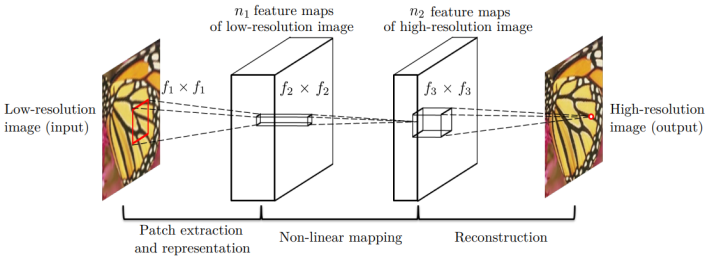


Figure 2: SRCNN network architecture.

An attempt of solving this computational problem was made by the same authors of SRCNN that produced another SR network called FSRCNN [7]. Differently from SRCNN, FSRCNN receives the non-upsampled LR image. Then, 5 layers compute the output image:

1. feature extraction: the first layer computes feature maps of d channels from the input LR image using
2. shrinking: to reduce the computational cost, instead of working on d channels, this 1x1 conv layer shrinks the number of channels to s < d channels to reduce feature depth
3. non-linear mapping: multiple convolutional layers are used to map LR feature maps to HR feature maps
4. expanding: another 1x1 convolution is deployed to expand the number of channels from s to d, performing an inverse operation with respect to the shrinking layer
5. deconvolution: the final layer performs a deconvolution to upscale the feature maps to the desired resolution and to combine them to produce the output image

This network reduces the cost by applying two main ideas:

- remove bicubic upscaling at the beginning to reduce the computational cost thanks to a lower resolution; the upscaling phase is moved at the very end of the network
- use shrinking and expanding layer to reduce the number of channels (and consequently the computational cost) allowing the non-linear mapping to work on less channels

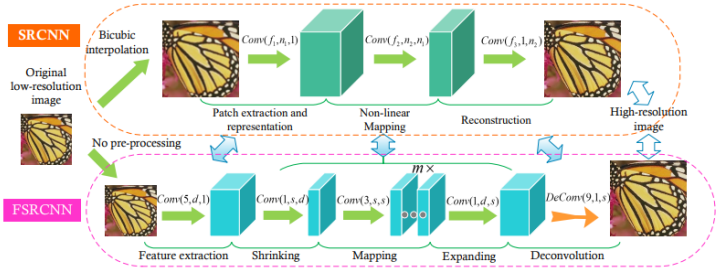


Figure 3: Comparison between SRCNN and FSRCNN architectures.

Another attempt of reducing the computational cost was made by Shi et al. [8] that created ESPCN network. Similarly to FSRCNN, this network increases the resolution of the feature maps at the very end of the network. The network is composed of 3 layers:

1. the first 2 layers are convolutional layers used to extract feature maps
2. the last layer is an efficient sub-pixel convolutional layer that upsamples the feature maps to produce the output image

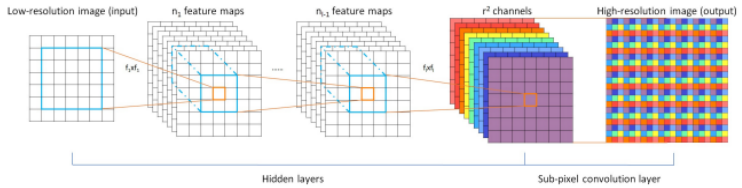


Figure 4: ESPCN architecture.

Moving upscaling at the end of the network allows us to use raw low resolution images, which also allows to use filter of smaller sizes to extract features, reducing the computational cost. This technique allows to reduce the whole computational cost and memory footprint of the network. If the network is not deep enough, however, the performances of it could be reduced, since the strength of deep learning shows up from deep network. SRCNN, FSRCNN and ESPCN are shallow networks because the authors were not able to train very deep versions of them due to training difficulties.

Kim et al. [9] introduced a SR network architecture called VDSR that allowed to train deeper networks. The network consisted in a stack of convolutional non-linear layer used to extract feature maps and map LR feature maps to HR feature maps. To train such a deep network, researchers added around it a residual connection going from the beginning to the end of the network. This network produces a residual image that is then added to the input image to reconstruct the output image. This allowed to increase the depth of the network, however the input image was, as with SRCNN, a bicubic interpolated LR image. Moreover, all layers use the same amount of output channels. The addition of the residual connection allowed to train a network of 20 convolutional layers, which is a huge improvement compared to the previous 3-layer networks.

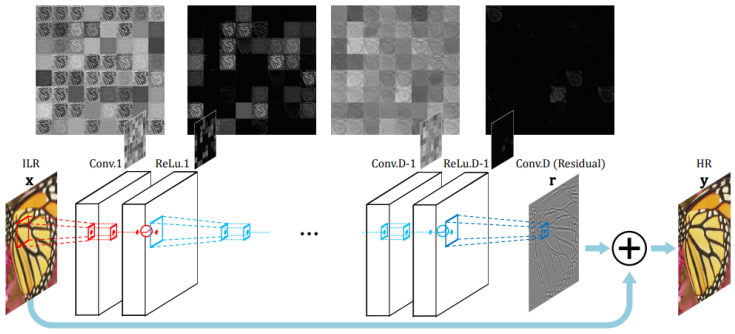


Figure 5: VDSR architecture.

Another huge improvement done in the training of a deep network for SR was done by Tai et al. [10], which created MemNet, a network composed of:

1. feature extractor: the first part of the network is a convolutional layer acting as a feature extractor
2. feature mapping: the second part of the network is a feature mapping part that maps LR feature maps to HR feature maps; in this network, this is a stack of a new block called "Memory Block", which is a block inspired by neuroscience that generates a long-term memory and is composed of two parts:

- the recursive unit, which learns multi-level representations which can be seen as short-term memory, composed of multiple recursive blocks; recursive blocks are residual blocks from ResNet
- the gate unit, which is a non-linear function allowing to maintain persistent memory

the short-term memory generated from the recursive unit and the long-term memory generated from the previous memory blocks are concatenated and sent to the gate unit through skip connections

3. reconstruction network: a final convolutional layer is used to reconstruct the residual image from the feature maps
4. as with VDSR, a residual connection from the beginning of the network to the end is used to add the input image to the residual image to create the SR image

Even though the network was still using a bicubic interpolated input image, this design allowed to train a network composed of a total of 84 layers.

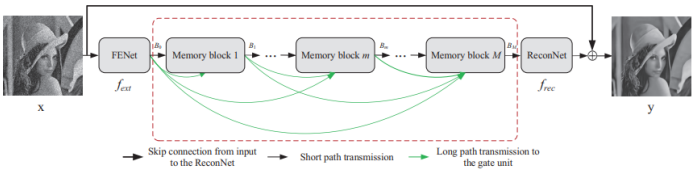


Figure 6: MemNet architecture.

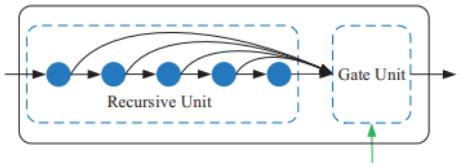


Figure 7: Memory Block from MemNet architecture.

Both VDSR and MemNet showed that deeper network with various types of skip connections achieve better performance with respect to the previous shallow networks, since skip connections help with the long-term dependency problem.

Lim et al. [11] created EDSR, a network consisting of:

1. feature extractor: a convolutional layer is used to extract features from the input image
2. residual module: a stack of residual blocks is used to map LR features to HR features; the used residual blocks are ResNet residual blocks without the last ReLU and without Batch Normalization; this stack of residual blocks is followed by a convolutional layer to compute the residual feature maps
3. a skip connection is used to add the feature maps extracted from the feature extractor to the residual feature maps
4. upsample module: this module consists of three parallel modules, each one of them working for, respectively, 2x, 3x and 4x images; each one of these paths consists in a conv layer followed by a pixel shuffle layer (the same layer used to upscale feature maps in ESPCN) which performs the actual upsampling; the 4x path consists simply in two stacked 2x modules
5. reconstruction module: a final convolutional layer is used to reconstruct the final SR image

As it is possible to see, this network is a mixture of the techniques previously illustrated:

- it uses raw LR image as input and upscales the feature maps at the end of the network, in order to reduce the computational cost of the network

- it uses a skip connection from the feature extractor to the upsample module, in order to train a deeper network

The residual module of EDSR contains 32 stacked residual blocks, resulting in a network deeper than MemNet.

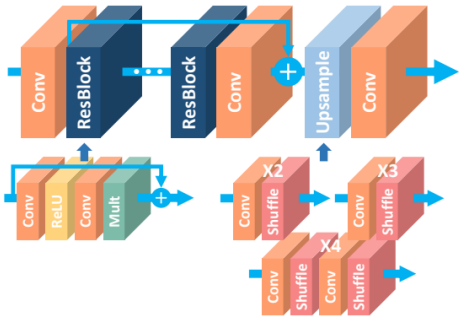


Figure 8: EDSR architecture.

Zhang et al. [12] introduced RDN, a network very similar to EDSR, having the only difference in the residual module. In fact, while having a global residual connection too, it consists in a stack of a new modified residual block called Residual Dense Block. The residual dense block has the following characteristics:

- consists in a stack conv+ReLU layers
- as with standard residual blocks, it has a local residual connection from the beginning to the end
- it has dense connections: the input of each convolutional layer in the block consists in a concatenation over the channel dimension of
  - the input of the whole block
  - the outputs of all the previous layers in the same block
- at the end of the block, all the outputs of the previous layers in the same block and the input of the block are concatenated along the channel dimension and passed to a 1x1 convolution that restores the depth of the feature map, in order to add the resulting feature map to the local residual connection

These dense connections produce an expansion of the feature map channels that produces an effect similar to the one produced in Memory Blocks from MemNet, in which the dense connections allow to fully use hierarchical features to produce a continuous memory of the previous states, allowing to learn multi-level representations. Similarly to the residual dense block, the whole residual module has a final concatenation of all the outputs of the previous residual dense blocks that is then passed to a final 1x1 convolution that restores the depth of the feature map, in order to add the resulting feature map to the global residual connection. The concatenation plus 1x1 convolutions at the end of the blocks and of the residual module act as a bottleneck to squeeze information in a reduced number of channels.

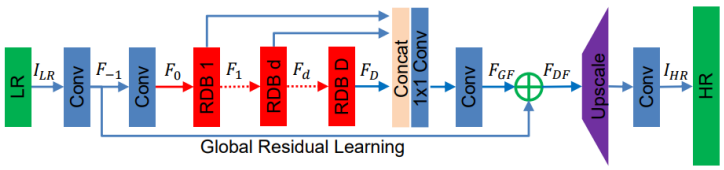


Figure 9: RDN architecture.

Li et al. [13] created MSRN, a network very similar to RDN that introduced a new type of residual block to use in the residual module. The introduced block is called Multi-Scale Residual Block and has the following characteristics:

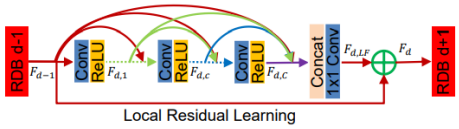


Figure 10: Residual Dense Block from RDN architecture.

- local residual connection
- a two-bypass structure: the block consists in two parallel branches (composed of two stacked conv+ReLU blocks); one branch is working with 3x3 convolutions while the other is working with 5x5 convolutions; the outputs of the first two convolutions in the two branches are concatenated and passed to the second convolutions of the two branches
- at the end of the block, the outputs of the second convolutions of the two branches are concatenated and fed to a 1x1 convolution restoring the feature map depth for the sum with the local residual connection, as in RDN

The structure of this multi-scale residual block is a mixture of a residual block and an inception block, in which the same feature map is fed to multiple parallel layers in order to produce various feature maps that are concatenated afterwards. This way, the information between those bypasses can be shared with each other so it is possible to detect the image features at different scales. This block allowed to train a network with more than 160 layers.

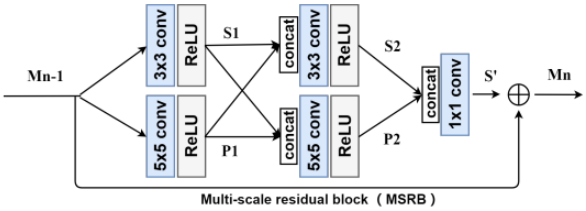


Figure 11: Multi-Scale Residual Block from MSRN architecture.

2.2 Lightweight Super Resolution Networks

Despite the fact that each network described in the previous chapter improved the PSNR scores of their predecessors and despite the fact that various improvements allowed to reduce the computational cost in most of the cases, these networks are not lightweight, in the sense that the number of parameters and total number of operations increased over time in order to achieve higher scores, however, the computational power of the machines on which they were trained increased too over time, so this was not a big problem. Nonetheless, this can lead to high risk of overfitting and, especially, limits for real world applications. SRCNN, FSRCNN and ESPCN were the first attempts of creating lightweight super resolution networks, but they were not performing well compared to deeper networks.

Ahn et al. [14] later created a network called CARN suitable for the mobile scenario implementing a cascade mechanism beyond a residual network. The network architecture is the same of RDN, but the residual module is changed to a stack of 3 Cascading Blocks and, instead of a global skip connections, all blocks are interconnected with global cascading (dense) connections: the input to a cascading block in the stack is the concatenation of all the output feature maps of previous concatenation block in the stack and the input of the whole stack. Moreover, a 1x1 convolution is used between cascading blocks to restore the number of channels in order to feed the map to the following cascade block. A Cascading Block is a stack of 3 group convolutions, each of them followed by a 1x1 pointwise convolution. This block comes from the idea of modifying a ResNet residual block by replacing standard convolutions with Group Convolutions, which are more efficient. Instead of a local skip connection, a Cascading Block uses local cascading connections, so the input of a conv layer is the concatenation along the channel dimension of all the outputs of the previous convolutional layers and of the input to the block. The 1x1 pointwise convolutions are thus used to restore the number of channels. In order to make the network even more efficient for mobile deployment, the group

convolutional layers used in the block share the same weights, namely they are the exact same layer, so that the block acts like a recursive layer.

This network thus uses some ideas already seen with previous networks:

- the use of dense cascading connections allows to learn multi-level representations (RDN)
- the use of group convolutions and recursive blocks (MemNet) allow to achieve a lower computational cost
- in order to make the network more lightweight, the convolutional layers in the block and blocks in the residual module are limited to a stack of only 3 layers/blocks

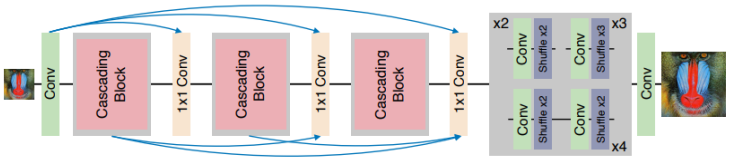


Figure 12: CARN architecture.

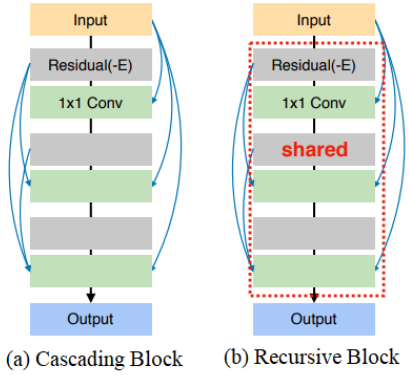


Figure 13: The cascading block (a) used in CARN. (b) shows the recursive strategy used to make CARN more efficient.

Neural Architecture Search based strategies were then used to construct more efficient networks, MoreMNAS [15] and FALSR [16], but due to limitations in strategy, the performance of these models is limited.

Luo et al. [17] introduced a network with a structure very similar to RDN but where the residual module consists in a stck of Lattice Blocks. These blocks are very similar to Multi-Scale Resudual Blocks from MSRN, so they consist in a two-bypass structure (also called two channel filter bank), where:

- each branch consists in a stack of 3 convolutional plus LeakyReLU layers
- the block does not have a local residual connection
- there are different feature map concatenations along the channel dimension
  - the input of the block is fed to the convolutional layer stack in the second branch and their output is concatenated to the input of the block itself to produce feature map A
  - A is fed to the convolutional layer stack of the first branch and the output is concatenated to A itself to produce a feature map B
  - B is concatenated with itself and fed to a 1x1 convolution that restores the number of channels to the input depth in order to feed the residual map to the next block

At the end of the residual module, all the output feature maps of the various lattice blocks are concatented along the channel dimension and fed to a Backward Fusion Module, which consists in:

- a stack of 4 lattice blocks that we are calling A, B, C and D
- the outputs of the 4 lattice blocks, that we are going to call A1, B1, C1 and D1, are fed to 1x1 convolutions that restore the number of channels avoiding them growing to produce the feature maps A2, B2, C2 and D2; these 4 feature maps are concatenated in the following way
  - D2 is concatenated to C2 to form feature map E1, which is then fed to a 1x1 convolution that restores the number of channels, producing a feature map E2
  - E2 is concatenated to B2 to form feature map F1, which is then fed to a 1x1 convolution that restores the number of channels, producing a feature map F2
  - finally, F2 is concatenated to A2 to produce the output of the BFM module

These modifications allow to achieve an economical structure that also achieves good results.

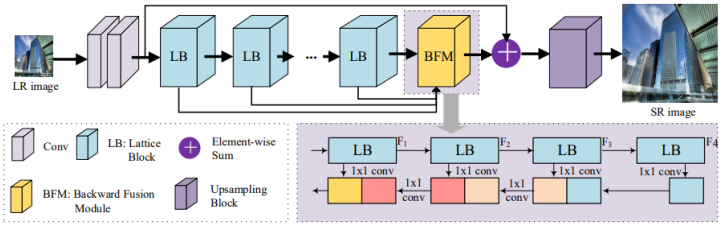


Figure 14: LatticeNet architecture.

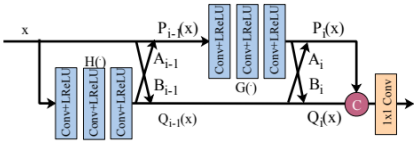


Figure 15: The lattice block used in LatticeNet.

All these works are suggesting that lightweight SR networks can keep a good trade-off between PSNR and parameters and, thus, computational cost of the network.

2.3 Super Resolution with Attention Mechanism

Attention mechanism is widely used in other Computer Vision tasks, achieving outstanding results. Attention can be seen as a guide to bias the allocation of available computational resources to the most informative element of an input. Woo et al. [18] proposed a Convolutional Block Attention Module, a block exploiting the inner-spatial and inner-channel relationship of features. This block is composed as follows:

- the input to the block is passed to a Channel Attention Module, producing channel attention features, working as follows:
  - two parallel pooling layers, a max and an average pooling, produce two feature maps A1 and B1
  - A1 and B1 are fed separately to a shared MLP producing two feature maps A2 and B2, that are summed together and fed to a sigmoid function to produce the output channel attention
- the channel attention is multiplied by the input feature map of the block and fed to a Spatial Attention Module, producing spatial attention features, working as follows:
  - two parallel pooling layers, a max and an average pooling, produce two feature maps that are concatenated together along the channel dimension to produce a feature map C
  - C is fed to a convolutional layer producing a feature map that is fed to a sigmoid to produce the output spatial attention

- the spatial attention is multiplied by the input feature map of the block to produce the output feature map of the block

This CBAM block is used in ResNet residual modulules at the end of it, so it is applied to the output of the convolutional layers and is used right before the sum with the residual connection. Even though this module was applied to ResNets for image classification, the ideas of Channel and Spatial attentions have been used by following papers, including the one described in this paper, working on super resolution.

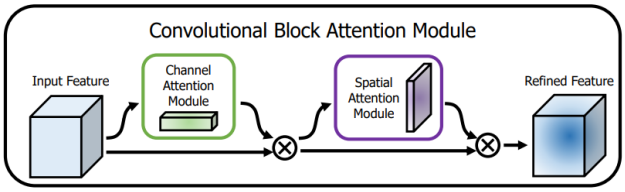


Figure 16: CBAM module.

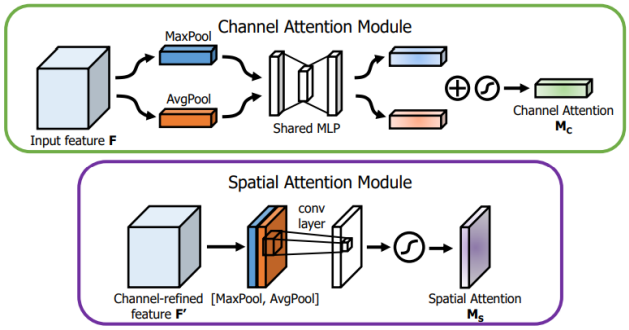


Figure 17: Channel and spatial attention modules used in CBAM module.

In this setting, recent works have introduced attention mechanism to networks for super resolution. Zhang et al. [19] introduced a very deep network called RCAN making use of channel attention mechanism, so considering only inner-channel informations. It is similar to RDB, but the residual module consists in a stack of Residual Groups, having only a global skip connection. Residual Groups are stacks of Residual Channel Attention Blocks and each Residual Group has only a local skip connection, while a Residual Channel Attention Block is a ResNet block without batch normalization and with channel attention ebedded after the convolutions. This channel attention consists in a stack of Global Average Pooling layer, Conv, ReLU and another Conv and the output of this stack is passed to a sigmoid and then multiplied with the input of the channel attention block. RCAB block representa a first-order attention network, that recalls first-order statistics.

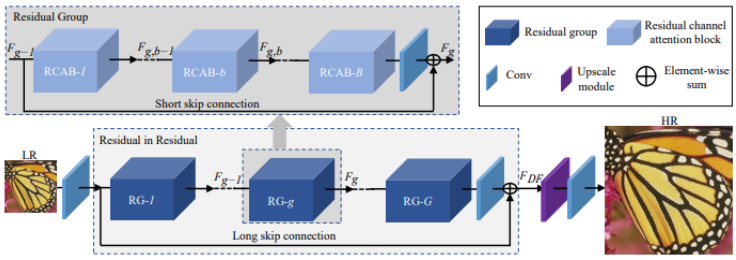


Figure 18: RCAN network.

Dai et al. [20] introduced a Second-Order Attention Network (SAN) which is similar to RCAN but the residual module is a stack of LSRAG modules. Moreover, skip connections are used to sum the input of the residual module to the output of each LSRAG module. LSRAG modules are similar to

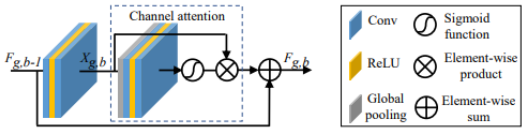


Figure 19: RCAB block used in RCAN.

ResNet blocks with channel attention that also embed a Second-Order Channel Attention block at the tail of the block, which is used to rescale inner-channel feature with their second-order statistics. Moreover, at the beginning and at the end of the Residual Module, a block called RL-NL, which exploits the abundant structure cues in LR features and the self-similarities in HR nature scenes

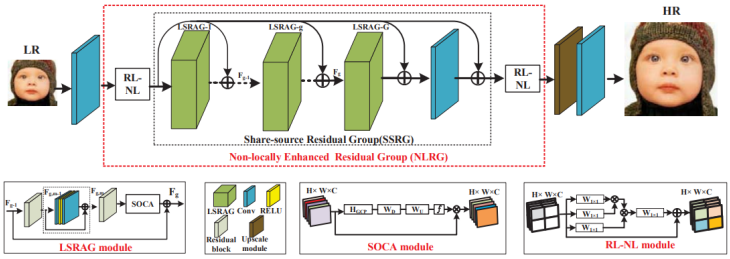


Figure 20: SAN network and its modules.

Hu et al. [21] introduced another attention-based network whose architecture called CSFM which is similar to previous networks but whose attention block is similar to CBAM but where spatial and channel attention are two separate parallel branches in the attention block. Both the Spatial Attention Unit and Channel Attention Unit use a residual connection.

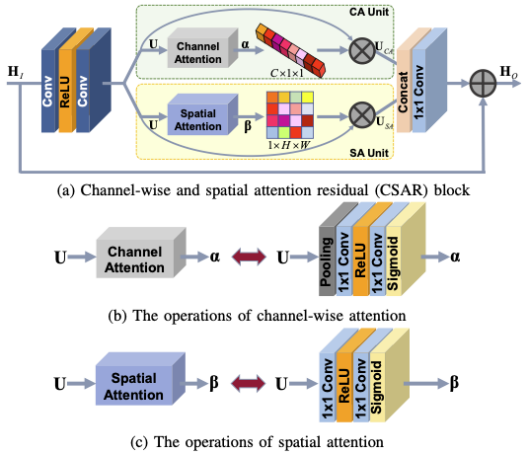


Figure 21: The CSAR attention block with channel and spatial modules used in CSFM network.

Li et al. [22] created RFANet, a network similar to RCAN where the residual module is a stack of RFA modules, a module consisting of:

- three residual blocks; after each residual block, the input of the RFA module is added to the output of the residual block through a skip connection
- after the 3 residual blocks, the resulting feature map is passed to a fourth residual block and the resulting output feature map is concatenated along the channel dimension to all the outputs of the previous 3 residual blocks
- the resulting feature map is passed to a 1x1 convolution to restore the number of channel and add it to the input of the RFA module through a local skip connection

- the residual blocks are not standard residual blocks, but rather a new block called Enhanced Spatial Attention (ESA) block, consisting of a conv + relu + conv stack, as with residual blocks, followed by an ESA mechanism which is a stack of a 1x1 convolution performing a channel reduction by a factor of 4, a strided convolution, a max pooling, a group convolution and a bilinear upsampling layer. The output of this stack is concatenated to the output of the first 1x1 convolution and then fed to another 1x1 convolution restoring the input depth; the output is then fed to a sigmoid and multiplied by the input of the ESA mechanism

This ESA mechanism reduces the channels of the feature map in order to make the ESA block lightweight. This ESA mechanism captures more powerful features with respect to standard spatial attention and forces the features to be more focused on the regions of interest. ESA block is an enhanced version of CSAR block.

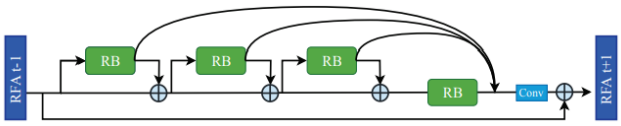


Figure 22: RFA module used in RFA network. Residual Blocks (RB) are replaced with ESA blocks.

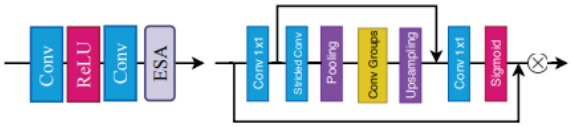


Figure 23: Left: the ESA block; right: the ESA mechanism used in the ESA block.

These networks prove that the use of an attention mechanism is also helpful for the Super Resolution task.

3 Multi-Path Residual Network (MPRNet)

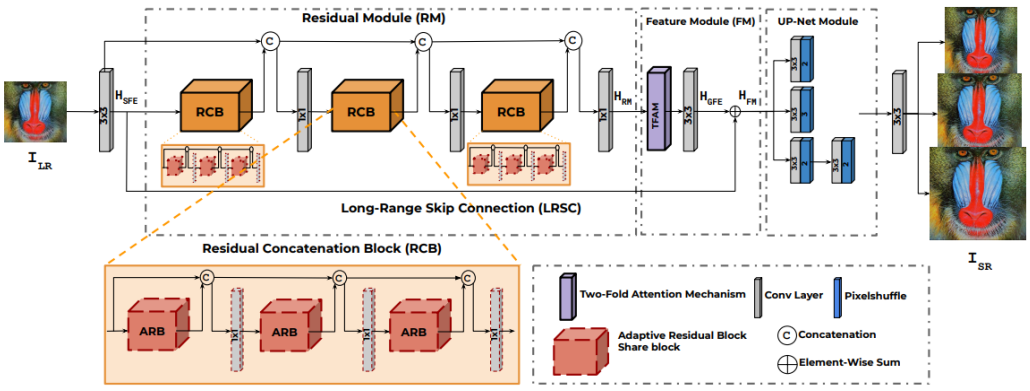


Figure 24: MPRNet architecture.

The architecture of MPRNet is basically a mixture of previous findings with some new ideas brought by the authors. This means that the overall outline of the network architecture is basically the same of EDSR, RDN, MSRN, CARN, LatticeNet, RCAN, SAN and RFANet, so the architecture basically consists in a stack of modules performing different functions.

3.1 Shallow Feature Extractor

The first module of the layer consists in just a single 3x3 convolutional layer which extracts initial features from the LR image.

3.2 Residual Module

As with previous architectures, the following module is the Residual Module, consisting in a stack of residual blocks. As with previous networks, the blocks used in this module are not standard ResNet residual blocks, but rather modified or improved blocks whose structure takes inspiration from the original ResNet residual block. This module is used to build a residual feature map that is then used in conjunction with the original feature map extracted by the Shallow Feature Extractor. As we saw with previous networks, this module can either be a stack of modules or a "nested" structure, consisting in a stack of modules which are, in turn, stack of modules. This is also the case for MPRNet Residual Module, which is a stack of Residual Concatenation Blocks, which are in turn stack of other blocks.

The Residual Module is thus a stack of three Residual Concatenation Blocks. The residual module receives as input the feature map computed by the Shallow Feature Extractor. Each Residual Concatenation Block receives a feature map and produces another feature map. The input and output of a RCB are concatenated, along the channel dimension, and fed as input to the following RCB, but before that they are fed to a 1x1 convolution layer which controls the depth of the feature map restoring the original input channel dimension, in order to feed the map to the next RCB. This is done before each RCB, but the input map that is concatenated to the output map is actually the input map that is obtained before the compression made by the 1x1 convolutional layer.

RCBs allow to learn multi-level representations, while the residual connections used to produce the concatenated feature maps help to propagate them through the network, while the 1x1 convolutions are used to maintain the computational expense of the network constant through the whole network (by keeping the depth of the feature map the same) without losing information. This module is build by following previous works that use skip connections and stacking to train deeper networks, while at the same time avoiding overfit and problems due to the depth of the network tanks to the skip connections. Moreover, they keep inspiration from lightweight networks using 1x1 convolutions between the blocks to avoid growing the channel depth through the network. The number of stacked blocks is kept to three to keep the network lightweight, as CARN. RCBs are, in turn, stacks of other blocks called Adaptive Residual Block (ARB). In particular, each RCB is a stack of three Residual Concatenation Blocks. As with the residual module, each block produces a feature map that is concatenated along the channel dimension with the input feature map and then passed as input to the following ARB after being passed to a 1x1 convolution that restores the depth of the feature map. Again, this sub-stacking is similar to the one done by previous networks, for example RFA or RCAN, which allows to train even deeper network by keeping computational cost constant. Again, the number of stacked blocks is kept to three to keep the network lightweight, as CARN; moreover, to keep the network lightweight, inside an RCB block the ARB blocks share the same weights, so namely they are implemented as a single ARB called three times (they are the exact same block). This is the same idea used in CARN, where the weights of the convolutional layers inside a Recursive Block where the same. This inner-stacking allows, again, to learn multi-level representation and propagate them through the network while at the same time keeping the computational constant thanks to 1x1 convolutions.

3.2.1 Adaptive Residual Blocks

The Adaptive Residual Block is a block designed to be an efficient and effective residual block based on depthwise and pointwise convolutions. To avoid losing information due to the narrow feature space produced by standard residual blocks, ARB proposed multiple learning pathways, more precisely three learning pathways, each of them being responsible of extracting different kind of information that are then aggregated together. This way, ARB blocks have access to more rich information.

The first path of an ARB is the Bottleneck Path: this path was designed to extract richer spatial information, since spatial information has key importance in the SR task, while at the same time preventing these feature maps growing the computational load in the middle of the path, so to maintain the whole path low-cost and efficient. For this reason, the beginning of the BN path is a depthwise convolution with a small kernel size (3x3), which allows to learn expressive features

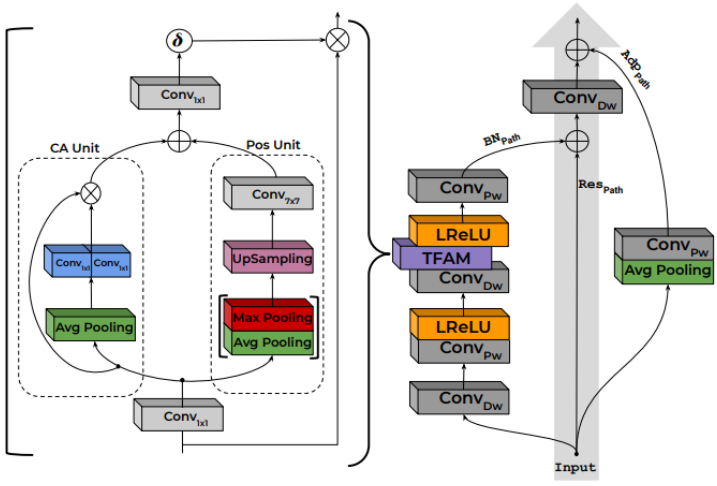


Figure 25: ARB block used inside RCB blocks in MPRNet (right). The TFAM module used MPRNet (left).

when applied to high dimensional space, which is our case, while at the same time being lightweight. Then, a pointwise convolution (a 1x1 convolution) is used to produce new features and reduce computational cost. After PW convolution, a LeakyReLU layer is used, followed by another 3x3 Depthwise convolution, followed by a new attention module called Two-Fold Attention module, which is used to spotlight the informative features along the channel and spatial axes. Finally, the bottleneck path is concluded with the use of another pointwise convolution layer. This path is desiged by taking inspiration from the bottleneck with inverted residual block used in MobileNetV2 [23]. Depthwise convolutions are nothing but group convolutions where the number of groups is the same as the number of channels, so a 2D kernel is learnt for each channel, and for this reason they are more lightweight than standard convolutions but, compared to it, the channel works on the same spatial and channel resolution, making the path efficient and low cost. For this reason, same spatial resolution and channel resolution are kept across the whole path.

In addition to that, the bottleneck path includes a new attention module called Two-Fold Attention Module, which is used to exploit spatial information and which takes some ideas from previous mentioned attention modules and improves them. The design of this block is similar to previous attention blocks with some additional modifications:

- similarly to the ESA mechanism of the RFA module used in RFANet, the first layer of the TFAM module is a 1x1 convolution performing a depth reduction by a certain reduction factor; this is actually not so clear in the paper, so we supposed this is the way TFAM was working, considering previous attention blocks very similar to TFAM from which the authors clearly took inspiration
- similarly to the CBAM module, two attention submodule are used after the first 1x1 conv layer that focuses on two different informations
  - the first is a Channel Attention unit, that focuses on channel information in order to recover edges; similarly to CBAM channel attention module, it consists in a Global Average Pooling layer that exploits first-order statistics, producing a 1D activation map, followed by two 1x1 convolutional layers, each of them working in parallel and working on half the channels of the input feature map, also producing half the output channels of the output feature map, which are then concatenated to produce the output feature map; this is done to make the unit more low-cost; after that, the output feature map is multiplied by the input feature map of the unit to generate the output channel attention map; the channel unit then modulates features globally by computing a summary feature for each channel thanks to global average pooling, channel attention unit emphasizes meaningful maps while redundant useless ones are diminished, focusing on what is meaningful given an input image;

  - the second is a Positional unit, that focuses on spatial information; it is similar to the Spatial Attention Module from CBAM, so it has a max pooling layer and an average pooling layer with a large kernel that compute two different feature maps that are then concatenated along the channel dimension; these two layers serve to obtain a large receptive field, which the positional unit needs for the SR task, so these two pooling have a large kernel size; after this layer, following the ESA attention, an upsampling layer is used to recover the spatial dimension, followed then by a 7x7 convolution layer to generate the output spatial attention map
  - while being very similar to CBAM attention modules, differently from this network the two attention modules are not stacked but they work in parallel, namely they work on the same input feature map and produce two different feature maps, the spatial and channel attention maps

- similarly to CSAR module, spatial and channel attentions are two separate and parallel branches
- the two output maps are then added together and fed to a final 1x1 convolution that restores the original channel depth, by expanding the number of channels from the reduced channels to the original channels.
- finally, the feature map is fed to a sigmoid function and then multiplied by the feature map that the whole TFAM block receives as input

The TFAM block is thus an attention module that works similarly to previously seen attention modules, like the CBAM attention and ESA attention, while at the same time introducing some improvements. As with previous works, the TFAM module is placed basically at the end of the bottleneck path.

The second path is a Residual Path, which basically is an identity function carrying the input feature map to then add it to the output feature map of the Bottleneck path. After this sum, the resulting feature map is passed through another depthwise convolution. This depthwise convolution is added to the residual path because authors find out it is crucial for the performance since it encourages the network to learn more meaningful spatial information.

The third path is called Adaptive Path, which consists in a global average pooling layer, which is used to take the average value of the feature maps to smooth and eliminate noise from LR image while, at the same time, reducing the dimensionality of each feature map, followed by another 1x1 pointwise convolution. The output feature map is then added to the feature map produced by the residual path to compute the output feature map of the whole ARB block.

3.3 Feature Module

The output feature map of the Residual Module is fed to a Feature Module, which is composed by another TFAM module which is used to re-calibrate (refine) the feature maps, followed by a 3x3 convolutional layer that is used to extract more abstract features. This final residual map is added, via a long-range skip connection, to the feature map produced by the Shallow Feature Extractor, to alleviate the gradient vanishing/exploding problem and make sure that the network has access to unmodified information.

3.4 Up-Net module

The final module is the up-net module which reconstructs the SR image from the obtained feature maps. This module is exactly the same of previous networks, so it has three branches for 2x, 3x and 4x upscale, consisting in a 3x3 convolution followed by a Pixelshuffle layer (for the 4x upscale, it consists in two stacked 2x modules).

3.5 Reconstruction module

The output feature map of the up-net module is fed to a 3x3 convolution which performs the actual reconstruction, as with previous networks.

4 Experimental Results

4.1 Settings

The model was trained using DIV2K [24] training set, consisting of 800 images. Each training batch was composed of patches of size 64x64 randomly extracted from 64 LR images, thus the batch size was 64. These patches are then augmented by random horizontal flips and random 90 degree rotations. AdamP was used as optimizer and the initial learning rate was set to 1 x 10ˆ3 and is halved every 4 x 10$\hat{5}$ training steps. L1 is used as loss.

The model was validated using DIV2K validation set, consisting of 100 images. The model was then tested using Set5 [25], Set14 [26], B100 [27] and Urban100 [5] datasets. The metrics used to evaluate and test the model where PSNR and SSIM, computed between the SR and ground truth HR image over the Y channel of the YCbCr color space, following previous works. Also following previous works, three different degradation models for the testing phase have been made.

The first degradation model consists only in a Bicubic Interpolation (BI) to downscale HR images and feed them to the network. This means that an LR image is downscaled via bicubic interpolation by a certain scale and then fed to the network that upscales it for the same scale. The output is compared to the original HR as described before. This is done on all the 4 test datasets for x2, x3 and x4 scales.

The second degradation model consists in applying a Gaussian blur with a 7x7 kernel and $\sigma = 1.6$ to the HR image, then, the blurred image is downsampled with bicubic interpolation by scaling factor x3. This model is called Blur-Downsampled model.

The third degradation model consists in downscaling the HR image with bicubic interpolation by scaling factor of x3 and then 30% of gaussian noise is added to the image.

4.2 Results

Params-MAC / Dataset	Methods / Scale	VDSR [18] $655K-612.6G$ PSNR/SSIM	LapSRN[20] $813K-149.6G$ PSNR/SSIM	MemNet[37] $677K-2662.4G$ PSNR/SSIM	NLRN[26] $350K-32.5$ PSNR/SSIM	SRFBN_S[23] $483K-119G$ PSNR/SSIM	CARN[1] $1592K-90.9G$ PSNR/SSIM	CBPN [48] $1197K-97.9G$ PSNR/SSIM	OISR_RK2_s[12] $1540K-114.2G$ PSNR/SSIM	MAFFSRN-L[32] $830K-38.6G$ PSNR/SSIM	LatticeNet[29] 777K-43.6G PSNR/SSIM	MPRNet [Ours] **538K-31.3G** PSNR/SSIM
Set5	×2	37.53/0.9587	37.52/0.9590	37.87/0.9597	38.00/0.9603	37.78/0.9597	37.76/0.9590	37.90/0.9590	37.90/0.9600	38.07/0.9607	**38.15/0.9610**	38.08/0.9608
	×3	33.66/0.9213	——	34.09/0.9248	34.27/0.9266	34.20/0.9255	34.29/0.9255	——	34.39/0.9273	34.45/0.9277	34.53/0.9281	**34.57/0.9285**
	×4	31.35/0.8838	31.54/0.8850	31.74/0.8893	31.92/0.8916	31.98/0.9594	32.13/0.8937	32.21/0.8944	32.21/0.8903	32.20/0.8953	32.30/0.8962	**32.38/0.8969**
Set14	×2	33.03/09124	33.08/0.9130	33.28/0.9142	33.46/0.9159	33.35/0.9156	33.52/0.9166	33.60/0.9171	33.58/0.9172	33.59/0.9177	33.78/0.9193	**33.79/0.9196**
	×3	29.77/0.8314	——	30.00/0.8350	30.16/0.8374	30.10/0.8350	30.29/0.8407	——	30.33/0.8420	30.40/0.8432	30.39/0.8424	**30.42/0.8441**
	×4	28.01/0.7674	28.19/0.7720	28.26/0.7723	28.36/0.7745	28.45/0.7779	28.60/0.7806	28.63/0.7813	28.63/0.7822	28.62/0.7822	28.68/0.7830	**28.69/0.7841**
B100	×2	31.90/0.8960	31.80/0.8950	32.08/0.8978	32.19/0.8992	32.00/0.8970	32.09/0.8978	32.17/0.8989	32.18/0.8996	32.23/**0.9005**	32.25/**0.9005**	**32.25**/0.9004
	×3	28.82/0.7976	——	38.96/0.8001	29.06/0.8026	28.96/0.8010	29.06/0.8434	——	29.10/0.8083	29.13/0.8061	29.15/0.8059	**29.17/0.8073**
	×4	27.29/0.7251	27.32/0.7280	27.40/0.7281	27.48/0.7306	27.44/0.7313	27.58/0.7349	27.58/0.7356	27.58/0.7364	27.59/0.7370	27.62/0.7367	**27.63/0.7385**
Urban100	×2	30.76/0.9140	30.41/0.9100	31.31/0.9195	31.81/0.9249	31.41/0.9207	31.92/0.9256	32.14/0.9279	32.21/0.8950	32.38/0.9308	32.43/0.9302	**32.52/0.9317**
	×3	27.14/0.8279	——	27.56/0.8376	27.93/0.8453	27.66/0.8415	28.06/0.8493	——	28.03/0.8544	28.26/0.8552	28.33/0.8538	**28.42/0.8578**
	×4	25.18/0.7524	25.21/0.7560	25.50/0.7630	25.79/0.7729	25.71/0.7719	26.07/0.7837	26.14/0.7869	26.14/0.7874	26.16/0.7887	26.25/0.7873	**26.31/0.7921**

Figure 26: Comparison of the model with previous lightweight SR models on BI degradation.

Regarding BI degradation model, MPRNet was compared with 10 most recent lightweight SOTA SR models for scale factors x2, x3 and x4. MPRNet achieves superior results especially well on Urban100, which contains rich structured contents and textures and MPRNet can consistently accumulate hierarchical features regarding these textures. MPRNet, thanks to residual connections, makes full use of features on all hierarchy levels, so it can generate more precise detail and reconstruct textures more correctly, as it is possible to see from Figure 27: in this figure, it is possible to see that the texture direction on scale x4 was reconstructed correctly on both images, differently from the other models. At the same time, it is the network with the lowest number of parameters and Multiply-Add operations for the x4 scale.

Regarding BD and DN degradation models, MPRNet is compared again with the 8 recent SOTA lightweight SR models, for scale factor x3. MPRNet achieves again remarkable results, surpassing RDN or reaching similar performances both on BD and DN models. It is also possible to see from Figure 29 that MPRNet does a better job in cleaning off noise and blurred regions compared to previous methods.

4.3 Ablation studies

Ablation studies were performed on TFAM and ARB modules and Residual Learning Connections.

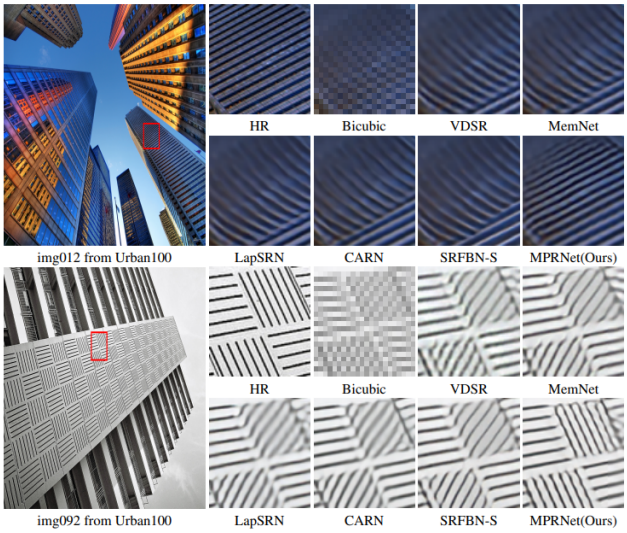


Figure 27: Qualitative results of MPRNet compared to previous models on BI degradation with x4 scale factor. Images from Urban100 [5].

Dataset	Methods / Degradation	Bicubic PSNR/SSIM	SPMSR[33] PSNR/SSIM	SRCNN[7] PSNR/SSIM	FSRCNN[8] PSNR/SSIM	VDSR[18] PSNR/SSIM	IRCNN_G[45] PSNR/SSIM	IRCNN_C[45] PSNR/SSIM	SRMD(NF)[39] PSNR/SSIM	RDN[47] PSNR/SSIM	MPRNet [Ours] PSNR/SSIM
Set5	**BD**	28.34/0.8161	32.21/0.9001	31.75/0.8899	26.58/0.8224	33.29/0.9139	33.38/0.9182	29.55/0.8246	34.09/0.9242	**34.57/0.9280**	**34.57**/0.9278
	DN	24.14/0.5445	—	27.04/0.7638	24.28/0.7124	27.42/0.7372	24.85/0.7205	26.18/0.7430	27.74/0.8026	28.46/0.8151	**28.54/0.8175**
Set14	**BD**	26.12/0.7106	28.89/0.8105	28.64/0.7997	24.86/0.7246	29.58/0.8259	29.73/0.8292	27.33/0.7135	30.11/0.8364	**30.53/0.8447**	30.47/0.8427
	DN	23.14/0.4828	—	25.56/0.6592	23.25/0.5956	25.60/0.6706	23.84/0.6091	24.68/0.6300	26.13/0.6974	**26.60/0.7101**	26.25/0.6954
B100	**BD**	26.02/0.6733	28.13/0.7740	27.33/0.7500	24.15/0.6728	28.61/0.7900	28.65/0.7922	26.46/0.6572	28.98/0.8009	**29.23/0.8079**	29.19/0.8062
	DN	22.94/0.4461	—	25.45/0.6198	23.95/0.5695	25.22/0.6271	23.89/0.5688	24.52/0.5850	25.64/0.6495	25.93/0.6573	**25.95/0.6616**
Urban100	**BD**	23.20/0.6661	25.84/0.7856	25.19/0.7591	22.95/0.6836	26.68/0.8019	26.77/0.8154	24.89/0.7172	27.50/0.8370	**28.46/0.8581**	28.31/0.8538
	DN	21.63/0.4701	—	23.59/0.6580	21.74/0.5724	23.33/0.6579	21.96/0.6018	22.63/0.6205	24.28/0.7092	24.92/0.7362	**25.00/0.7406**

Figure 28: Comparison of the model with previous lightweight SR models on BD and DN degradation.

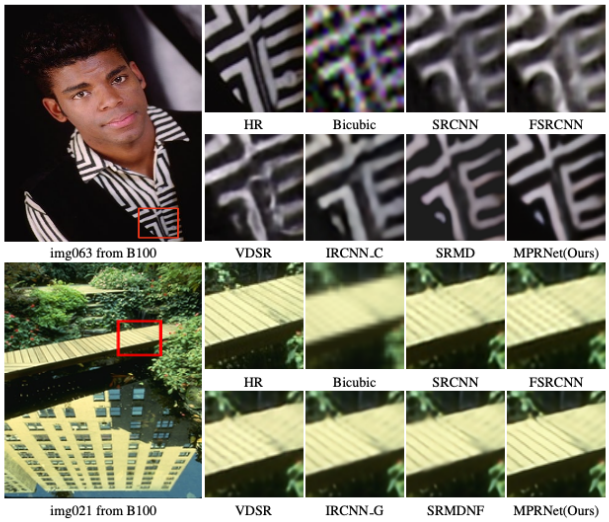


Figure 29: Qualitative results of MPRNet compared to previous models on BD and DN degradation with x3 scale factor. Images from B100 [5].

4.3.1 Attention

Regarding the use of attention mechanisms, two different ablation studies were performed.

The first ablation study consists in taking three of the previously mentioned networks not using an attention mechanism, namely EDSR, RCAN and MSRN, and then adding three attention modules at the end of their blocks to to test the effectiveness of attention in SR tasks. The tested attention blocks were CSAR [21] attention, ESA attention (used in RFANet) and, finally, TFAM used in MPRNet. Networks were all trained with their default settings. As it is possible to see from Figure 30, all baseline models are increased and PSNR is improved when applying CSAR to EDSR and MSRN, but doesn't show any improvement to RCAN. On the other hand, ESA helps in all settings since it is an improved version of CSAR, even though it does not use spatial informations, so TFAM was introduced, which considers both channel and spatial informations. TFAM shows better performances compared to all previous attention blocks. Table 3 also shows the efficiency of replacing residual blocks of these three networks with ARB using TFAM, showing that using ARB and TFAM together improves even more the results with large margin.

Name	EDSR					RCAN					MSRN				
	Baseline	Attention Modules			ResBlock	Baseline	Attention Modules			ResBlock	Baseline	Attention Modules			ResBlock
Channel and spatial attention residual[16]		✓					✓					✓			
Enhanced Spatial Attention[27]			✓					✓					✓		
Two-Fold Attention Module[Ours]				✓	✓				✓	✓				✓	✓
Adaptive Residual Block[Ours]					✓					✓					✓
PSNR on Set5 (×4)	32.46	32.48	32.51	32.54	**32.65**	32.63	32.64	32.67	32.70	**32.78**	32.25	32.27	32.30	32.34	**32.39**
PSNR on Urban100 (×4)	26.64	26.66	26.69	26.71	**26.79**	26.82	39.84	26.86	26.89	**26.96**	26.22	26.25	26.29	26.32	**26.41**

Figure 30: Ablation studies over EDSR, RCAN and MSRN using CSAR, ESA and TFAM.

The second ablation study on attention consists modifying MPRNet attention mechanism in ARBs and Feature Module by replacing TFAM with recent attention modules, namely SE (squeeze-and-excitation), CBAM, CSAR, ESA and TFAM. As it is possible to see from Figure 31, CBAM and SE do not improve the baseline (model without attention), and this is due to the fact that channel information is lost due to spatial attention being applied sequentially after channel attention and spatial attention using a max pooling layer. On the other hand, CASR achieves better result due to the use of the two attentions in parallel and not one after another, so considering both, but again as in the previous experiment, since ESA is an improvement of CSAR, so it performs better. TFAM again performs better among all attention modules, because channel attention and spatial attentions are computed in parallel and due to CA Unit using a global average pooling to compute first order statistic and Pos unit using max pooling operations, which allow to detect most important spatial features like edges.

Dataset	Baseline	SE	CBAM	CSAR	ESA	TFAM
Set14 (×4)	28.57	28.59	28.54	28.61	28.64	**28.67**
Urban100 (×4)	26.19	26.21	26.18	26.23	26.25	**26.29**

Figure 31: Ablation studies over MPRNet using SE, CBAM, CSAR, ESA and TFAM.

4.3.2 Adaptive Residual Block

Another ablation study was performed on the ARB block. MPRNet was modified and trained in the same setting while at the same time ARB blocks in the networks were replaced swith residual blocks from SOTA models, namely BottleneckBlock from MobileNet, ResBlock from EDSR, RCAB block from RCAN and of course ARB block from MPRNet. As it is possible to see from Figure 32, BottleneckBlock could not perform well in SR task due to inability of extracting high-frequency information. ResBlock performs slightly better but still has difficulties in extracting rich feature maps and eliminating noises from LR space. RCAB block performs better due to presence of channel attention, but ARB block showed even better performances due to the ability of the block to extract more expressive spatial information also thanks to TFAM's channel and spatial attentions, while having access to high dimensional information. In the same figure, it is possible to see that researchers also did ablation studies on ARB paths, by training a model with ARBs with only the bottleneck path (ARB_B), with bottleneck and adaptive paths (ARB_BA), with bottleneck and residual paths (ARB_R) and with all paths (ARB). It is possible to se that the model using blocks with all paths is

the one achieving best performances among all tested residual blocks and pathways. This is because with the use of all three pathways, the network is forced to focus on more informative components and more abstracted features of the LR space. In fact:

- residual path makes information propagate locally
- adaptive path adaptively extracts informative features
- bottleneck path extracts more meaningful spatial information

This way, different levels of the network access to rich and more expensive feature maps.

Configs	MobileNet BnBlock	EDSR ResBlock	RCAN ResBlock	$\mathbf{ARB}_B$	$\mathbf{ARB}_{BA}$	$\mathbf{ARB}_R$	ARB
BN_p				✓	✓	✓	✓
Adp_p					✓		✓
Res_p						✓	✓
B100 (×4)	27.24	27.44	27.52	27.46	27.58	27.55	**27.63**
Urban100 (×4)	25.79	25.96	26.08	26.05	26.15	26.11	**26.31**

Figure 32: Ablation studies over MPRNet using BottleneckBlock, ResBlock, RCAB and ARB blocks.

4.3.3 Residual Learning Connections

Last ablation study was performed on Local Residual Connections (LRC), i.e. residual connections between ARB blocks in a RCB block, Global Residual Connection (GRC), i.e. residual connections between RCB blocks in the Residual Module, and Long-Range Skip Connection (LRSC), i.e. the residual connection going from the Shallow Feature Extractor all the way to the end of the Feature Module. MPRNet using only GRC performs slightly better than the baseline (network without connections), since GRCs help transporting information from mid to high layers, leveraging multi-level representations. MPRNet using only LRC could not perform better than the same model using only GRC, due to the fact that 1x1 conv on the residual connection can confuse optimization, but using both LRC and GRC connections shows better performances, due to GRC easing the issue of information propagation of LRC. Finally, LRSC helps because it overcomes the vanishing gradient problem, so the network with all connections performs better than networks with only some connections.

Options		Baseline	$1st$	$2nd$	$3rd$	$4th$	$5th$
Residual Learning Connections	LRC	✗	✓			✓	✓
	GRC	✗		✓		✓	✓
	LRSC	✗			✓		✓
PSNR on Set5 (×3)		34.42	34.40	34.47	34.45	34.52	**34.57**
PSNR on Urban100 (×3)		28.30	28.29	28.35	28.33	28.38	**28.42**

Figure 33: Ablation study on MPRNet using LRC, GRC and LRSC residual connections.

4.4 Network Complexity Analysis

Finally, authors performed a network complexity analysis by comparing their model with 15 recent SOTA SR models. As it is possible to see from 34, MPRNet achieves best performances among lightweight SR networks in terms of PSNR, while keeping a good trade-off between the number of parameters and performances.

5 Project work

The project work consisted in a Python implementation of MPRNet from scratch in PyTorch, while at the same time executing the same tests with the same degradation models done by the authors on Set5 [25], Set14 [26], B100 [27] and Urban100 [5] SR benchmark datasets. Finally, the model was also tested with the same degradation models on three additional SR testing sets, namely Manga109 [2, 3], SunHays80 [4] and historical dataset.

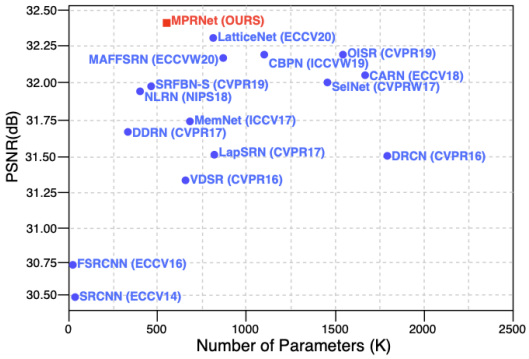


Figure 34: Network complexity analysis.

5.1 Implementation details

The network was implemented in Python using PyTorch library, by following the paper. Regarding this, the paper was not 100% clear on some aspects of the architecture, especially regarding hyper-parameters, but since MPRNet is basically a puzzle of previous network parts that work well plus some simple ideas, it was possible to reconstruct the network architecture and hyperparameters by exploring the literature. In addition to the implementation of the model, we developed Dataset and DataLoader classes to load datasets as training, validation and test sets. A training set is loaded by extracting and randomly augmenting LR and HR patches from the original LR and HR images, a validation set is instead loaded as a standard dataset, with LR and HR full images as the input and the ground truth. Finally, datasets are loaded as testing sets by applying the degradation models described in the paper to produce LR images to feed to the network, starting from HR images which will also serve as ground truth. We also developed training and testing loops and, to make training and testing experiments less painful, we used Hydra library, which allows to read a .yaml configuration files such that we don't need to modify the source code to start a new experiment but only the configuration file and/or the CLI command, since Hydra allows to override configuration parameters in configuration files when executing the code via Python's CLI interface.

The network was developed using PyCharm in conjunction with Git and GitHub. In addition to PyTorch, standard Python libraries like Pillow, numpy, scipy, scikit-image etc. have been used. The model was trained and tested on Google Colab, using its free GPU. We also used Wandb to log metrics and image outputs for qualitative and quantitative results and comparisons. The project source code is available on GitHub [28], while all the training, validation and testing results are available on Wandb's project page [29].

5.2 Training

Regarding training, not all the hyperparameter values were disclosed by the authors, so in order to train our network we reviewed the literature and picked common and reasonable values from previous works when unavailable in the original paper.

The network was trained with DIV2K [24] training set using a batch size of 64 and LR patches size of 64×64, where each batch consists of 64 LR and corresponding HR patches all corresponding to the same scaling factor, so a single batch has patches from a single scaling factor. Used scaling factors are x2, x3 and x4, so our training set consists in automatically generated batches that can be from these three scaling factors. The patches are also augmented with random horizontal flips and random 90° rotations. The optimizer used was AdamP with an initial learning rate of 1×10^{-3} that is halved every 2000 training steps, where a step is the processing of a single batch, up to minimum possible learning rate of 1×10^{-4}.

Differently from the paper, we halved the learning rate earlier because using higher learning rates for longer times was producing spikes in the training/validation losses, as a consequence of the exploding gradients problem, that were sometimes halting the training itself and setting the network on a wrong learning path, meaning that the training loss and metrics were only getting worse after these spikes. This was of course producing the same effects on validation loss and metrics too. To mitigate the

exploding gradients problem, we also added gradient clipping with value 10, but it was not enough. So we still kept a high learning rate but we greatly reduced the halving steps, boosting the training at the beginning while slowing it down afterwards. The problem with this approach is that if we keep halving the learning rate this fast, we will soon end up having a very low learning rate that will make the training very slow since the beginning, so we introduced a minimum learning rate of 1×10^{-4} such that lower learning rates are rounded up to this value, in order to keep the training at an acceptable speed that was, at the same time, not producing the exploding gradients problem. This solved the exploding gradients problem and we didn't observe spikes in the loss anymore. An example of these training spikes can be seen in Figure 35.

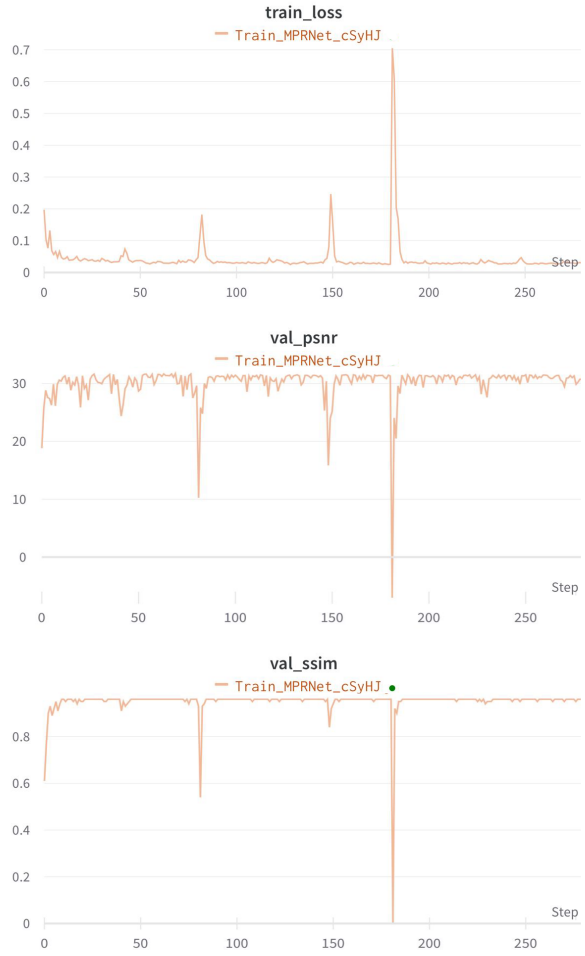


Figure 35: An example of training loss spikes due to exploding gradients and their effect on validation metrics ("steps" are epochs).

The network was trained for a total of 6×10^4 training steps using L1 loss. The authors did not disclose the number of training steps they used to train their model, however by looking at previous works we could deduce that these kind of SR networks are usually trained for around 6×10^5 training steps. Nonetheless, we used Google Colab's free GPU to train the network, so we were not able to train the network for this long because the available GPU was slow and we also had a limited notebook execution time so, by using checkpoints, we were able to train the network for around one week up to 1/10th of the (alleged) original training steps.

Authors also did not disclose the depth, i.e. the number of channels, of the feature maps across the networks. Authors did however specify that the feature map depth was kept constant across the whole

network, so by looking at previous SOTA SR methods, the most used value for this parameter was 64, so we decided to set feature maps depth to 64.

Another aspect of the network architecture that was not clear in the paper was if the TFAM attention module was reducing the depth of the feature maps inside the module, to then expand them to the input depth at the end of the module like in previous attention modules. By looking at the literature, it is possible to see that most attention blocks, especially the ones similar to this TFAM from which the authors took inspiration, all reduced the feature map depth in the attention module by a certain reduction factor. We then implemented the TFAM module this way, but since authors were also not disclosing the used reduction factor, we again drew from the literature. Previous attention-based networks were using a reduction factor of 16, 4 or 2., so we decided to use 16 as reduction factor inside TFAM attention modules because most of previous networks were using 16 as reduction factor when using a feature map size of 64 across the network. Moreover, by setting reduction factor to 16, we were able to obtain the same number of trainable parameters in the x4 branch that was indicated by the authors in Figure 26, indicating the fact that this might actually be the value actually used by the authors for this hyperparameter.

Another unspecified parameter was the kernel size of average and max pooling layers in TFAM's Positional Unit. Authors only said that it was a "large kernel" and, by looking at the literature, we found out previous works using similar attentions were using a 7x7 kernel, so we used the same value.

Table 1 shows all the values used for the hyperparameters.

Training hyperparameters	
Training steps	60000
Batch size	64
Patch size	64×64
Feature map depth	64
TFAM reduction factor	16
TFAM Pos unit pool kernel	7×7
Patches augmentations	Random horizontal flips & 90° rotations
Training scale factors	x2, x3, x4
Starting learning rate	0.001
Learning rate halving steps	2000
Minimum learning rate	0.0001
Optimizer	AdamP
$\beta 1$	0.9
$\beta 2$	0.999
ϵ	0.00000001
Loss function	L1 loss
Gradient clipping value	10
Validation hyperparameters	
Validation scale factors	4
Reproducibility seeds	
Random seeds	1507

Table 1: Hyperparameters used for the training and testing.

5.3 Validation

Regarding validation, we evaluated the network during training using DIV2K validation set after every training epoch. However, we decided to use only 5 images from this validation set, selected in order to have both detailed and more simple images, because feeding 100 images after each epoch would greatly slow down the training. On top of that, we only used one scale factor for the validation phase, namely x4 scale, because using different scale factors for the validation set after each epoch would have produced a very fluctuating validation loss, an effect that can be seen with the training loss due to batches in the same epoch using different scale factors. We decided to use x4 scale factor since it is the most difficult to restore, so it gives us a better sense of how and if the network is improving. The paper wasn't reporting which scale factors were used for the validation phase and if they were using the full dataset, but because of our limited computing power we decided to validate

this way, also following previous works in the literature. We didn't consider 3x and 2x scales since they are easier than 4x and usually produce a higher score, meaning that 4x is enough to indicate the quality of the model.

We didn't use only training and validation losses to evaluate the model but also training and validation PSNR and SSIM metrics, as the authors did.

5.4 Testing

For testing, we performed the exact same tests done by the authors, so we tested the model on Set5 [25], Set14 [26], B100 [27] and Urban100 [5] SR benchmark datasets. We tested out network using all the degradation models described by the authors. The LR images that are upsampled by the network are produced by three different degradation models that degrade HR images in the testing sets in three different ways:

- BI model: LR images are produced by downsampling the HR image with bicubic interpolation by a factor of 2x, 3x and 4x
- BD model: LR images are produced by applying a Gaussian blur with a 7x7 kernel and $\sigma = 1.6$ to the HR image, then, the blurred image is downsampled with bicubic interpolation by a factor 3x.
- DN model: LR images are produced by downsampling the HR image with bicubic interpolation by a factor of 3x and then 30% of gaussian noise is added to the downsampled image

The model was also tested with the same degradation models on three additional SR benchmark sets, namely Manga109 [2, 3], SunHays80 [4] and historical dataset, in order to test the model on datasets that have a different nature from the ones used by the authors and, eventually, observe any difference in results. In fact:

- Manga109 consists in a dataset of manga book covers, so it is mostly consisting of images with few details, flat colors, Japanese and English texts in form of titles and anything related to mangas and manga drawings
- SunHays80 consists in a dataset of natural and city landscapes that might include animals and humans, however animals and humans are not the main subject of the picture
- historical consists in a dataset of few, precisely 10, black and white low resolution historic images

5.5 Results

5.5.1 Training and validation

	MPRNet		bicubic	
DIV2K training set	best PSNR	37.17		
	best SSIM	0.9564		
DIV2K reduced validation set	best PSNR	27.87	PSNR	25.85
	best SSIM	0.9118	SSIM	0.8695
DIV2K full validation set	PSNR	28.38	PSNR	26.79
	SSIM	0.882	SSIM	0.8459

Table 2: Quantitative results of MPRNet and bicubic upsampling over DIV2K train and validation sets. Results are computed over x2, x3 and x4 upsampling for training set, while on x4 upsampling for validation set. Moreover MPRNet PSNR and SSIM scores for training and reduced validation set are the best achieved scores during training.

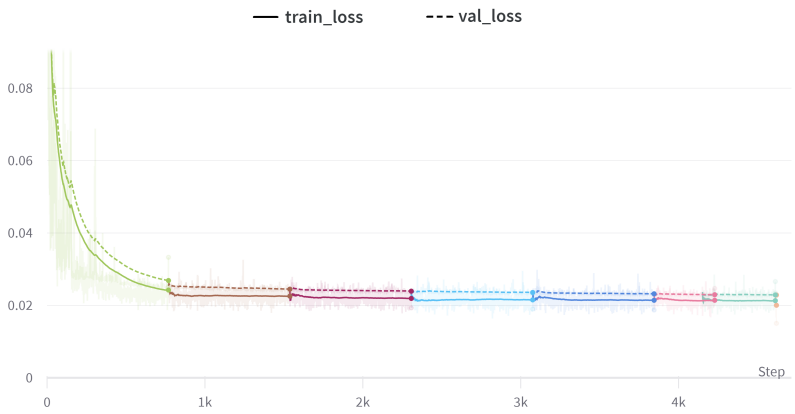


Figure 36: Training and validation loss compared. Different colors are for different parts of the training. Graph has been smoothed and zoomed for visualization purposes. "Steps" are epochs.

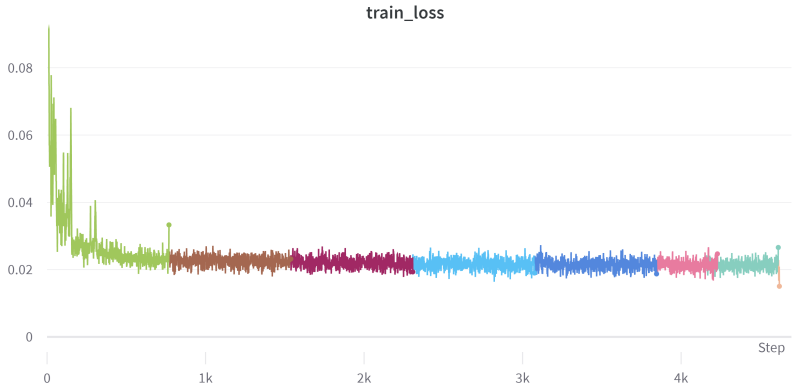


Figure 37: Training loss. The graph is zoomed for visualization purposes. "Steps" are epochs.

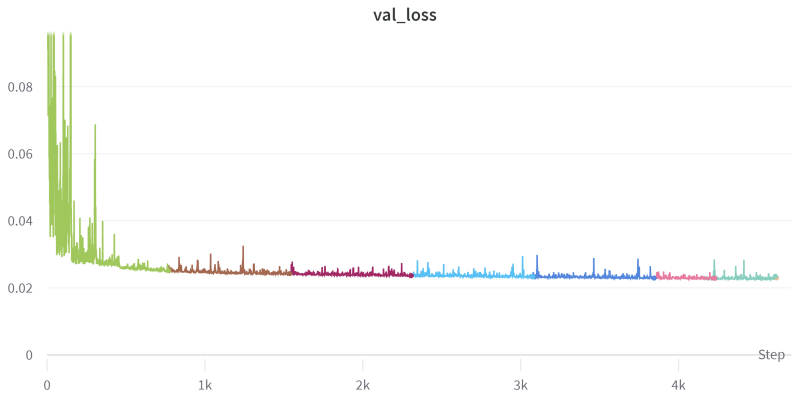


Figure 38: Training loss. The graph is zoomed for visualization purposes. "Steps" are epochs.

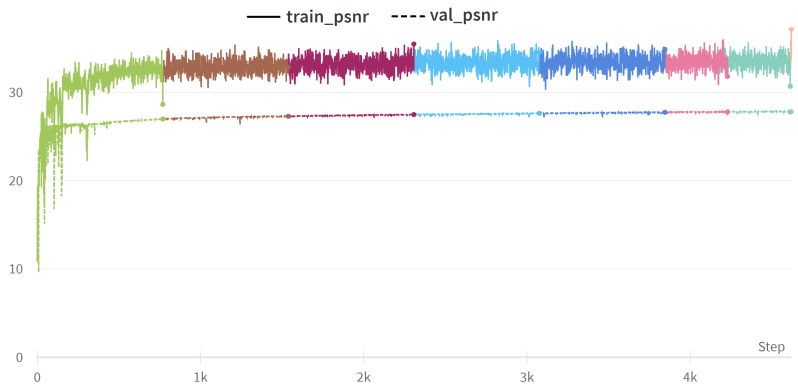


Figure 39: Training and validation PSNR. "Steps" are epochs.

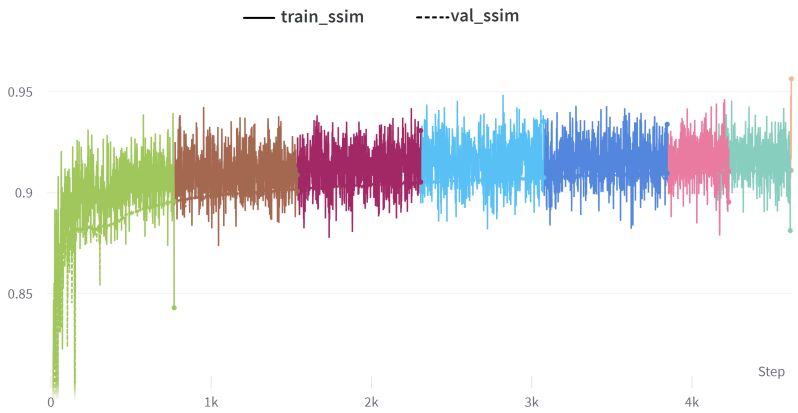


Figure 40: Training and validation SSIM. The graph is zoomed for visualization purposes. "Steps" are epochs.

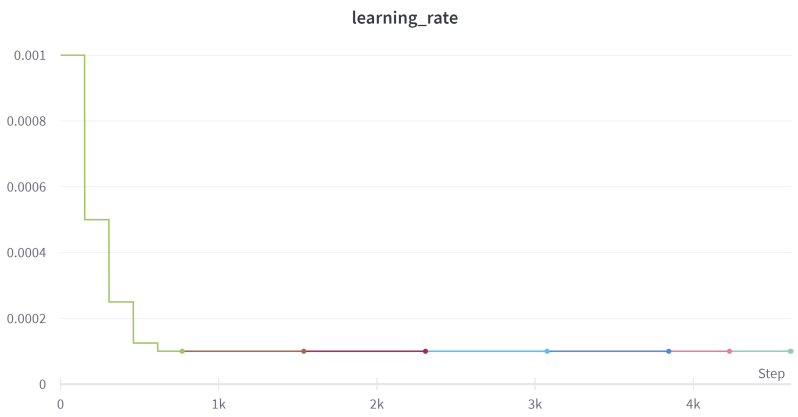


Figure 41: Learning rate during training.

Figure 42: 4x upsample on validation image from div2k using bicubic.

Figure 43: 4x upsample on validation image from div2k at 10k steps.

Figure 44: 4x upsample on validation image from div2k at 60k steps.

Figure 45: HR ground truth of validation image from div2k.

5.5.2 Testing

		MPRNet					Bicubic				
		BI			BD	DN	BI			BD	DN
		x2	x3	x4			x2	x3	x4		
Set5	PSNR	35.95	32.67	29.68	28.55	19.34	32.35	29.56	27.11	27.37	22.82
	SSIM	0.9776	0.9621	0.9268	0.9133	0.4733	0.9593	0.928	0.8771	0.8881	0.6566
Set14	PSNR	31.37	28.54	26.25	25.77	19.73	28.75	26.45	24.53	24.87	22.47
	SSIM	0.9443	0.9005	0.8436	0.8399	0.4896	0.9153	0.8614	0.7967	0.8108	0.6464
B100	PSNR	30.41	27.86	25.81	25.8	20.75	28.24	26.31	24.66	25.06	23.14
	SSIM	0.9295	0.8745	0.8102	0.8123	0.5273	0.8947	0.835	0.7689	0.7845	0.6614
Urban100	PSNR	29.11	26.47	23.54	23.37	18.8	25.54	23.87	21.81	22.45	20.99
	SSIM	0.9371	0.8901	0.8112	0.8046	0.4865	0.8837	0.8253	0.7414	0.7656	0.6148
Manga109	PSNR	36.91	30.69	27.65	25.25	18.81	29.95	25.92	23.95	23.89	21.46
	SSIM	0.9858	0.9572	0.9228	0.8903	0.473	0.9592	0.9037	0.8526	0.8555	0.6297
SunHays80	PSNR	33.89	30.97	28.55	28.67	19.88	31.73	29.36	27.33	27.77	23.32
	SSIM	0.9635	0.9271	0.8774	0.881	0.4468	0.9429	0.8993	0.8454	0.8579	0.6356
Historical	PSNR	27.57	24.22	22.23	22.27	19.01	25.34	22.83	21.23	21.48	20.89
	SSIM	0.9327	0.8535	0.7681	0.7597	0.5526	0.8867	0.7933	0.7026	0.7143	0.6525

Table 3: Quantitative results of MPRNet and bicubic upsampling over Set5, Set14, B100, Urban100, Manga109, SunHays80 and Historical using BI, BD and DN degradation models.

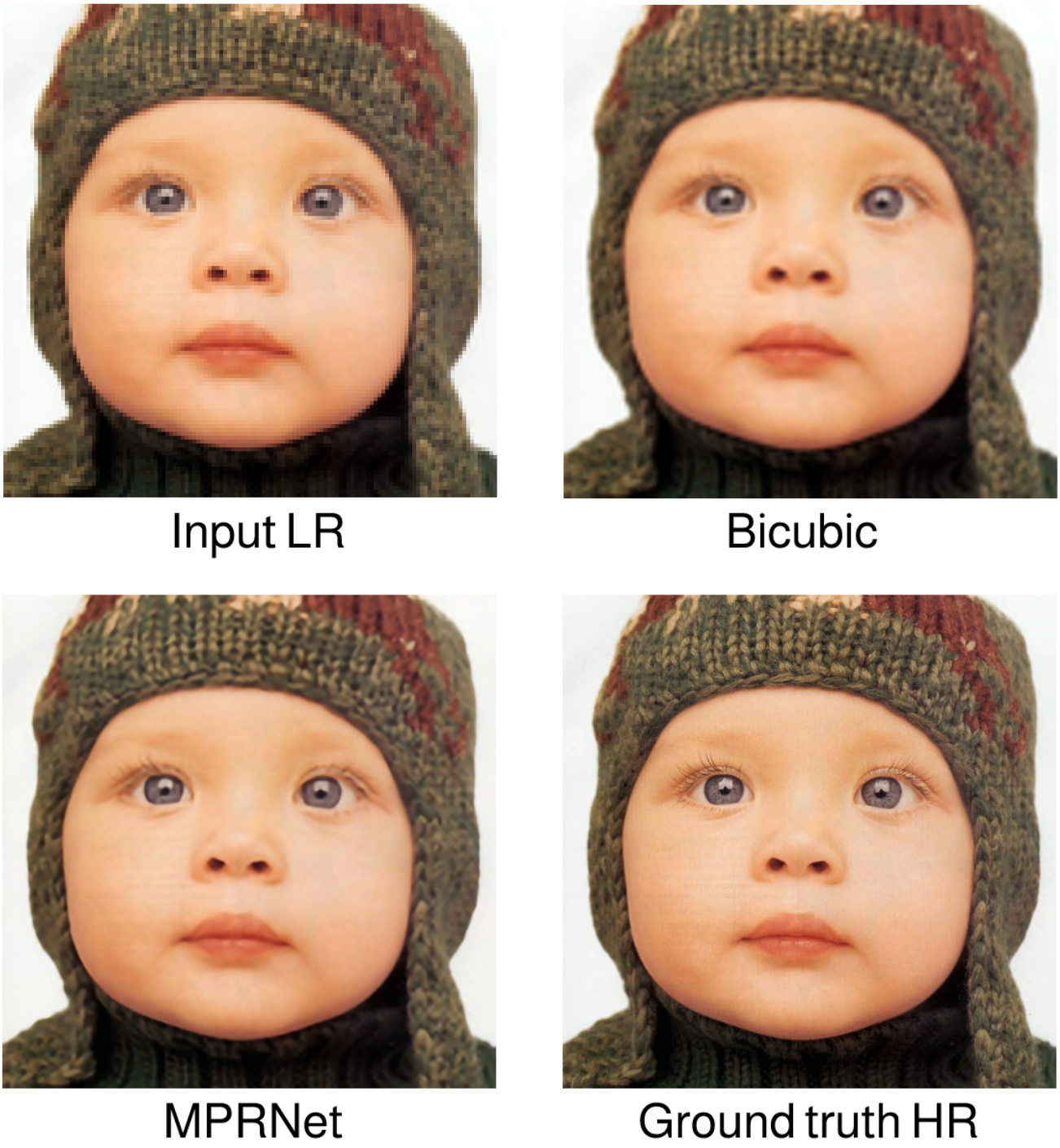
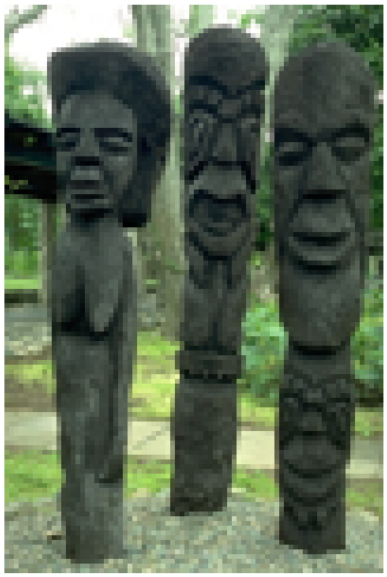


Figure 46: Qualitative result of bicubic and MPRNet x4 upsampling on x4 BI degraded Set5.

Figure 47: Qualitative result of bicubic and MPRNet x4 upsampling on x4 BI degraded Set14.



Input LR

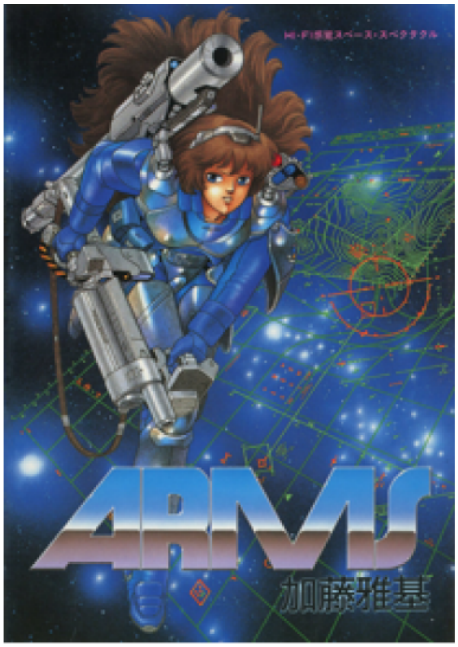
Bicubic

MPRNet

Ground truth HR

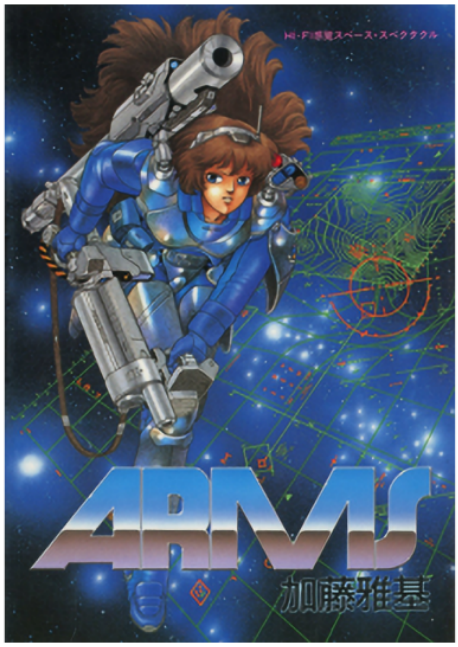
Figure 48: Qualitative result of bicubic and MPRNet x4 upsampling on x4 BI degraded BSD100.

Figure 49: Qualitative result of bicubic and MPRNet x4 upsampling on x4 BI degraded Urban100.

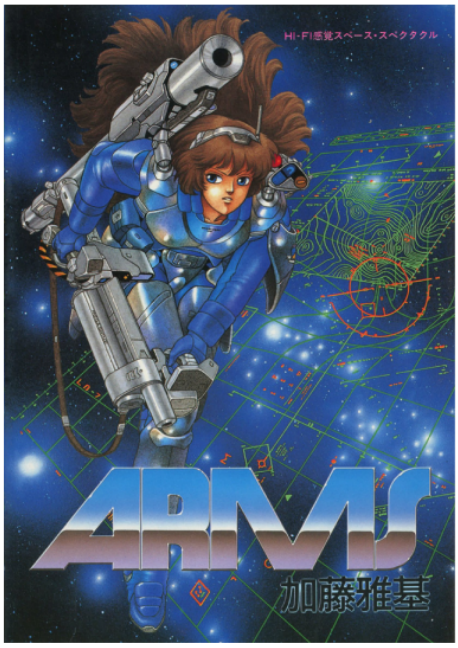


Input LR

Bicubic



MPRNet



Ground truth HR

Figure 50: Qualitative result of bicubic and MPRNet x4 upsampling on x4 BI degraded Manga109.

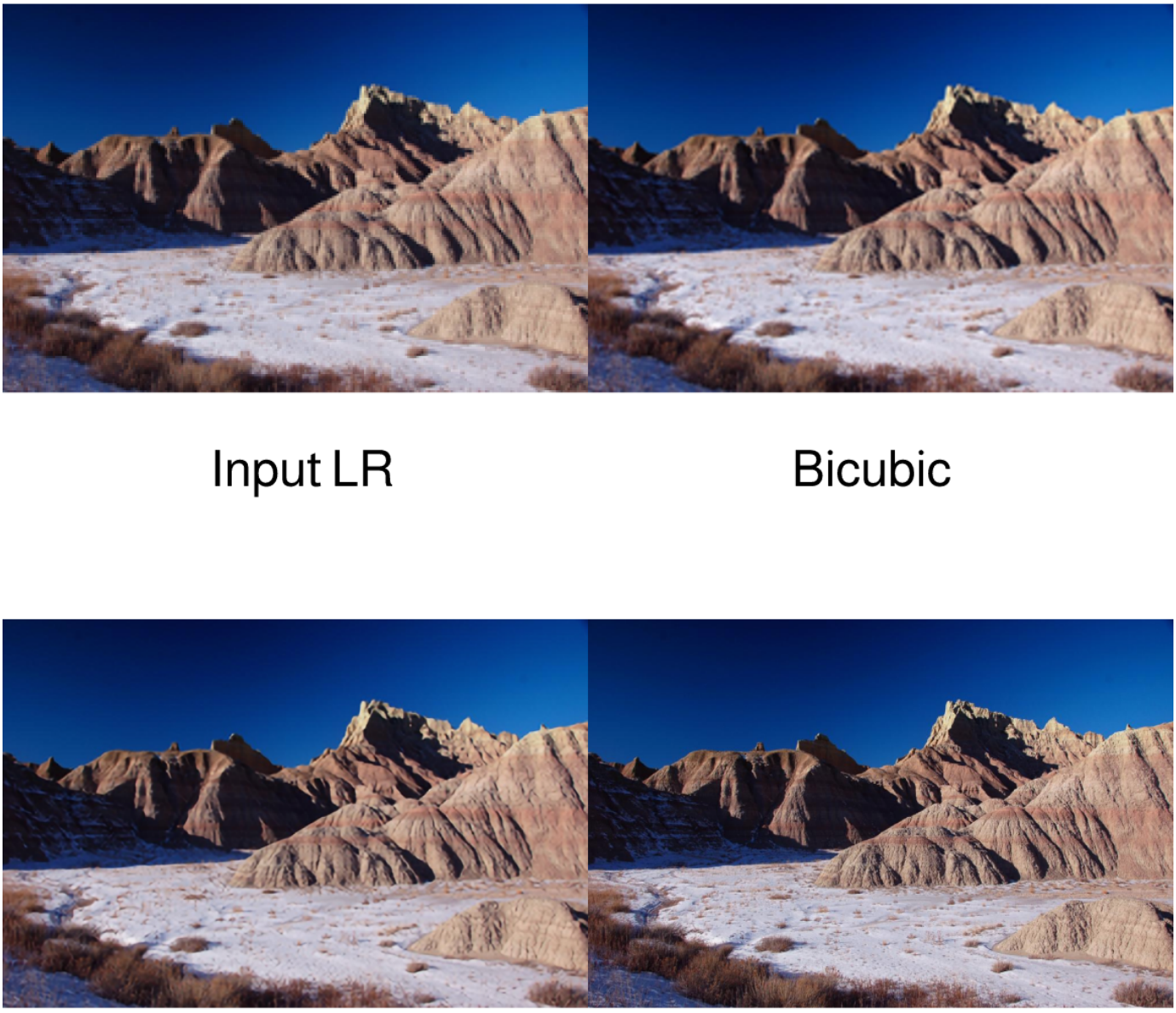


Figure 51: Qualitative result of bicubic and MPRNet x4 upsampling on x4 BI degraded SunHays80.

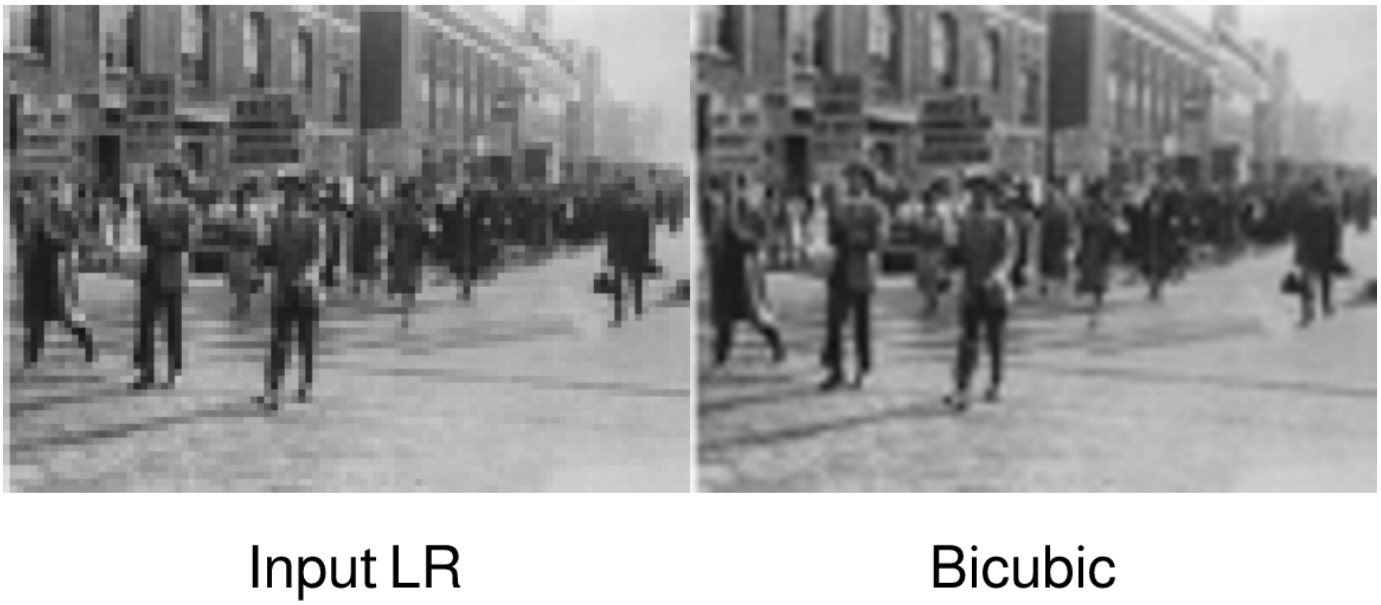


Figure 52: Qualitative result of bicubic and MPRNet x4 upsampling on x4 BI degraded historical.

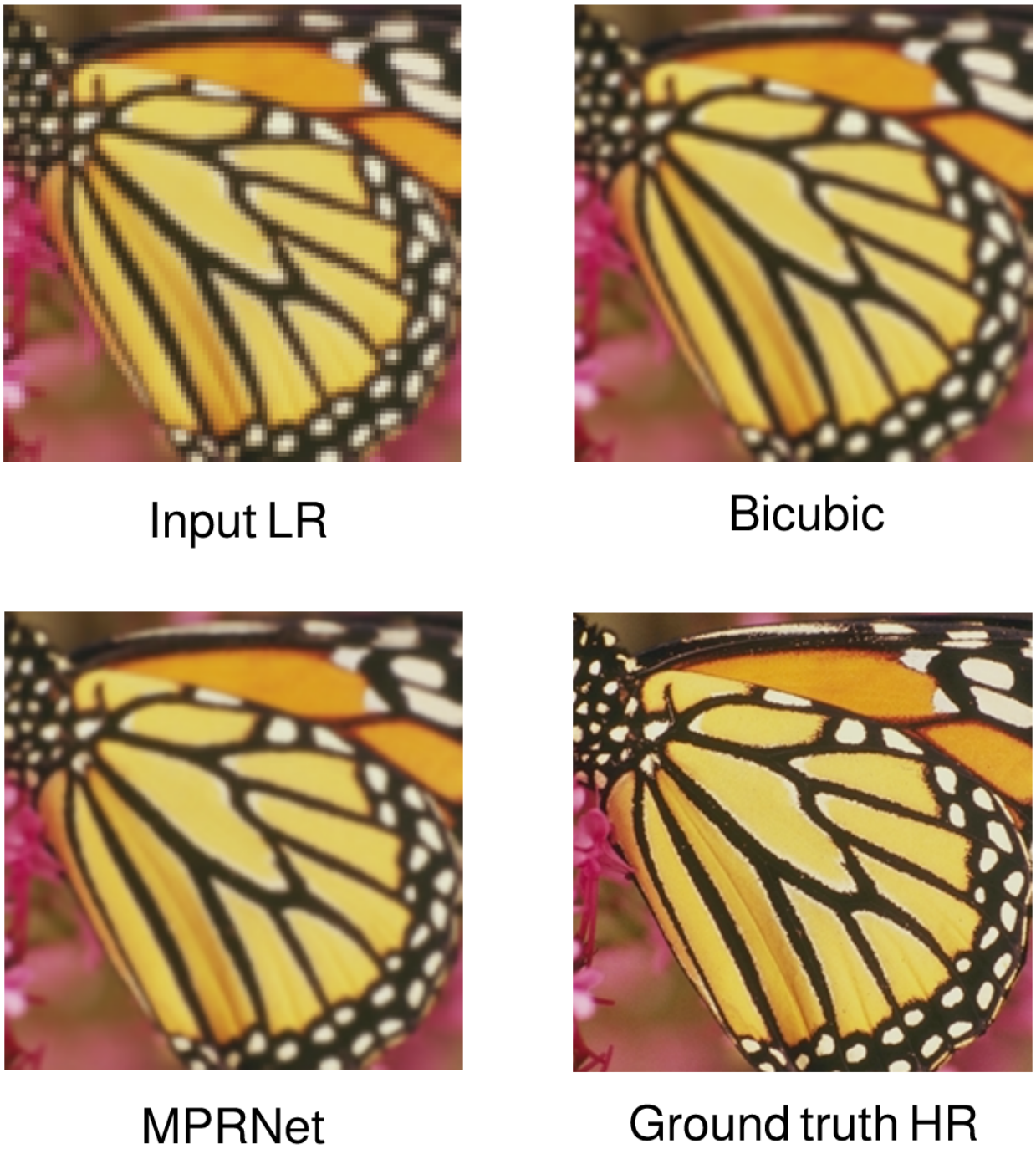


Figure 53: Qualitative result of bicubic and MPRNet x4 upsampling on BD degraded Set5.

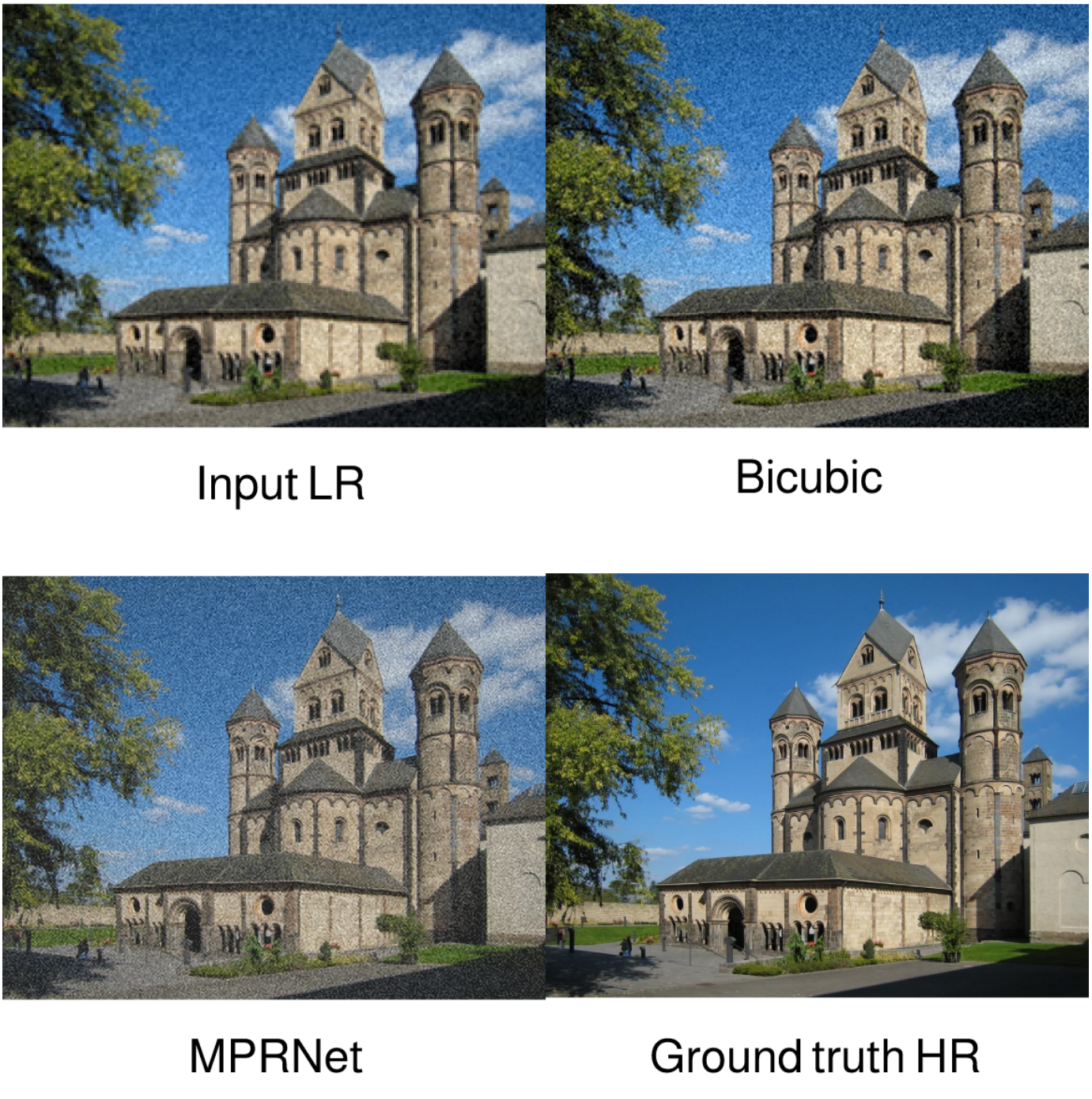


Figure 54: Qualitative result of bicubic and MPRNet x4 upsampling on DN degraded SunHays80.

5.5.3 Discussion about results

During training, we could observe a smooth descent in the training and validation losses. We didn't observe any abnormal peaks in the losses thanks to the reduced learning rate and the gradient clipping. As it is possible to see from Figure 36, the network shows no overfitting nor exploding gradient symptoms, indicating that an appropriate learning rate strategy helped counteracting that problem. From Figure 41 it is possible to see the learning rate with its step strategy from a max of 0.001 to a min of 0.0001 over the various training steps. However, it is possible to see from Figure 37 that the training loss is very fluctuating because of the various batches using different scales. On the other hand, from Figure 38 we can see that validation loss is not that fluctuating cause we used a single scale for all the samples in the dataset. Each color stage corresponds roughly to 20k training steps, for a total of roughly 60k training steps.

We can also see from Figure 39 and Figure 40 that a behaviour similar to the loss can be seen for PSNR and SSIM training and validation metrics. We can see a smooth ascent in both training and validation metrics, with no sudden drops due to exploding gradients, however we still see the fluctuation of the training metrics due to the same motivation explained for the loss.

We can see, however, that the losses are still slowly descending and that both PSNR and SSIM training and validation metrics are still increasing, indicating that there's still room for more training steps and more learning.

Regarding quantitative results for the training and validation phases, we can see that our model achieves a better maximum PSNR and SSIM score over the reduced DIV2K validation set (the one composed of 5 images) with respect to the score achieved by bicubic upsampling on the same set, with a difference of around 2% for PSNR and 5% for SSIM. The same behaviour can be seen for metrics computed on the whole DIV2K validation set, we can see that our model achieves a 2% higher PSNR score and a 4% SSIM score.

Regarding qualitative results for the training and validation phases, we can see how the model is improving over time by looking at Figure 43 and Figure 44, which are the output of the model on a validation image from DIV2K for scale factor 4x at, respectively, 10k and 60k training steps, compared to 45 which is the ground truth HR and compared to 42, which is the same upscaling done with bicubic. We can see a little bit more blurring in the 10k steps image and less defined edges. Edges in the early phase of the training show a more pixelation effect, similar to the one saw in standard bicubic upsampling, because the network has not yet learnt to smooth out the upsampling done internally by the network. The 60k steps image, however, still has edges a bit blurred and lacks of detail compared to the HR ground truth but, on the other hand, it is possible to see how even the 10k image produces an image of higher quality compared to the one produced by bicubic upsampling.

All these factors together are good indicators of the fact that the network is learning and improving the image quality over time. Moreover, the network is learning to produce and reconstruct sharper edges, but still struggles in reconstructing details and textures, mostly because, compared to the model shown in the paper, it has been trained for less steps. This is also evident in the tests we did after training.

Regarding test results, there are a few considerations to make by looking at quantitative results in Table 3 and at the qualitative results illustrated by Figures 46, 47, 48, 49, 50, 51, 52, 54 and 53.

We can see that our model performs better than bicubic upsampling in almost all tests: in fact, it is possible to see how PSNR scores are consistely about 2-3% higher for MPRNet, while SSIM scores are about 3-6% higher. By looking at Figures 46, 47, 48, 49, 50, 51 and 52, in fact, we can see how our model, as in validation, produces images with sharper edges and less blurry effect, however it still struggles in reconstructing the textures and the details in areas that present very complex textures and not close-to-flat colours. This effect is mitigated with lower scale factors, because the upscaling will be less difficult, but still we could see room for improvement mainly in texture and detail reconstruction.

Our model also performs better than bicubic upsampling on BD degradation: in fact, it is possible to see that PSNR scores for MPRNet are around 1% higher while SSIM scores are 2-4% higher. This behaviour is also visible in Figure 53, where it is possible to see how the image produced by our model still produces, as with BI tests, images that are less blurred especially on edges. The

network is still not able, though, to clear all the Gaussian blur in the LR image because scores for BD degradation are still lower than scores for 3x BI degradation.

The only test in which bicubic upsampling outperforms MPRNet is the test with DN degradation: in fact, we can see how this is the only column in the table in which all bicubic scores are greater than MPRNet scores by 2-3% for PSNR and by 10-20% for SSIM. We believe that the reason for this behaviour is due to the fact that the network has still not learnt to restore details and textures by ignoring the noise in the image. On top of that, we believe that being this a network that is not trained for image denoising, the network will never be able to perfectly clear the image from the noise, even if trained for the amount of steps indicated by the authors. Nonetheless, it will still be able to produce better images with respect to bicubic, as it is possible to see from the results on this test reported by the authors. The reason why our network does not produce good results over this degradation method, we can see in Figure 54 how our network actually super-resolves also the noise, producing an output image which shows a more defined noise too. On the other hand, the blurring nature of bicubic produces an output image whose noise a bit less visible because the blurring effect blended it better with the background image.

Regarding new sets, namely Manga109 [2, 3], SunHays80 [4] and historical sets, we can see that our model achieves the highest score for x2 BI degradation on Manga109 dataset across all sets. For the other degradation models, the scores of MPRNet are not the highest across all datasets but are still higher than the scores for Set14, B100 and Urban100 sets for mostly all tests. The reason for this is that, compared to the latter mentioned sets, Manga109 is composed of manga covers that generally present less texture and more flat colors, meaning that our network will struggle less to produce higher quality images because the textures to restore are simply less. The network is however able to reconstruct also text, as it is possible to see from 50, better than bicubic upsampling and reaches higher scores compared to bicubic over all tests except for, again, DN degradation.

Regarding SunHays80 set, our model performs similarly to how it performs on previous sets like Set5, Urban100 and Set14, and this is again because of the presence of more elaborated texures. Bicubic results on DN are, however, still higher that MPRNet results on the same degradation. Scores on Manga109 and SunHays80 are, however, among the highest over all sets, also because images in this set have higher resolution compared to the other sets, and for this reason the downsampled LR images will be of higher resolution compared to previous sets and will thus have more detail in the LR image, easing a little the upscaling process. This fact can be deduced also by the results on the same sets achieved by bicubic interpolation, which show a similar behaviour.

The importance of resolution is also confirmed by the scores achieved by MPRNet and by bicubic upsampling on historical dataset. These are, in fact, the lowest scores across all datasets. The main reason for this is that HR images in historical are LR images, so the downsampling process will produce even smaller images. Moreover, being these pictures historical ones, textures in the HR image are more blurry and there's less detail in general, so the network (but also bicubic) struggles to restore the quality of the image because the LR image is very low resolute and also has poor quality. By looking at Figure 52, in fact, we can see how the image produced by our model only has sharper edges compared to the one produced by bicubic upsampling, while textures are very blurry and basically absent in both images in the same way. Text is also not readable in both images and again this is because of the combination of the poor quality HR image and the low resolute and low quality HR image in the original set.

We are only reporting visual results on 4x BI degradation over all the testing sets in order to save space on this report, however we could observe a similar behaviour also on x2 and x3 BI degradations, and this is confirmed by the quantitative results. We are reporting only 4x upscalings also because this way we are able to see more the results of the upscaling produced by our network on the images, because with 4x upscaling the network will be forced to produce more details and to extract more informations from the textures for the production of the upscaled textures, so the effects of the network upscaling will be more visible on the output image compared to lower scale upscalings. For the same space reason, we are reporting only one visual result over BD degradation and over DN degradation, but still we can see a similar behaviour over the other datasets. More images for visual comparisons for the other tests (and the same tests) are available on Wandb project page [29], in which it is also possible to see the HR patches produced from the training set during training.

6 Conclusion

To conclude, we can see that our MPRNet is able to fully reconstruct edges, producing sharp and close-to-perfect edges, but still struggles in reconstructing textures and details from the LR image. It is possible to see, however, that the images produced by our model are still of highest quality than the images produced by bicubic upsampling, even though there's still space for improvement and even though results can be improved with a longer or complete training. In fact, the network struggles on datasets having images with complex textures and rich details, like Urban100, while performs best on datasets having less textures and more flat images, like Manga109. On top of that, the network produces better results when the HR images that are degraded have high resolution and are in general more detailed, such that the LR image will have more detail because of the higher resolution and of the more detailed image. On the other hand, the network struggles the smaller HR images are and the less detailed they are, because downsampling these images will compress the already few details available in the image, producing a LR image that has small space for improvement. An example of this situation is historical dataset. However, a high starting resolution is not always the answer to problems because if the textures in these images are very complex, the network will still struggle to reconstruct them, but it will still produce a SR image having more details than the image produced by upscaling the same image at a lower resolution. The difficulty of our network to restore textures is also confirmed by the results over DN degradations, where our model performs even worse than bicubic, indicating the fact that the network is able to increase the resolution of LR images without losing sharpness, but at the same time has poor ability of discerning the textures from the noise. To counteract these problems, the best way is to train the network longer and tune the hyperparameters better. Also, it is possible to implement a cosine learning rate scheduler to have a boost in training not only at the beginning.

References

[1] A. Mehri, P. B. Ardakani, and A. D. Sappa. Mprnet: multi-path residual network for lightweight image super resolution, 2020. DOI: 10.48550/ARXIV.2011.04566. URL: https://doi.org/10.48550/arxiv.2011.04566.

[2] Y. Matsui, K. Ito, Y. Aramaki, A. Fujimoto, T. Ogawa, T. Yamasaki, and K. Aizawa. Sketch-based manga retrieval using manga109 dataset. *Multimedia Tools and Applications*, 76(20):21811–21838, 2017. DOI: 10.1007/s11042-016-4020-z.

[3] K. Aizawa, A. Fujimoto, A. Otsubo, T. Ogawa, Y. Matsui, K. Tsubota, and H. Ikuta. Building a manga dataset "manga109" with annotations for multimedia applications. *IEEE MultiMedia*, 27(2):8–18, 2020. DOI: 10.1109/mmul.2020.2987895.

[4] L. Sun and J. Hays. Super-resolution from internet-scale scene matching. In *2012 IEEE International Conference on Computational Photography (ICCP)*, pages 1–12, 2012. DOI: 10.1109/ICCPhot.2012.6215221.

[5] J.-B. Huang, A. Singh, and N. Ahuja. Single image super-resolution from transformed self-exemplars. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5197–5206, 2015. DOI: 10.1109/CVPR.2015.7299156.

[6] C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks, 2015. DOI: 10.48550/ARXIV.1501.00092. URL: https://doi.org/10.48550/arxiv.1501.00092.

[7] C. Dong, C. C. Loy, and X. Tang. Accelerating the super-resolution convolutional neural network, 2016. DOI: 10.48550/ARXIV.1608.00367. URL: https://doi.org/10.48550/arxiv.1608.00367.

[8] 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, 2016. DOI: 10.48550/ARXIV.1609.05158. URL: https://doi.org/10.48550/arxiv.1609.05158.

[9] J. Kim, J. K. Lee, and K. M. Lee. Accurate image super-resolution using very deep convolutional networks, 2015. DOI: 10.48550/ARXIV.1511.04587. URL: https://doi.org/10.48550/arxiv.1511.04587.

[10] Y. Tai, J. Yang, X. Liu, and C. Xu. Memnet: a persistent memory network for image restoration, 2017. DOI: 10.48550/ARXIV.1708.02209. URL: https://doi.org/10.48550/arxiv.1708.02209.

[11] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee. Enhanced deep residual networks for single image super-resolution, 2017. DOI: 10.48550/ARXIV.1707.02921. URL: https://doi.org/10.48550/arxiv.1707.02921.

[12] Y. Zhang, Y. Tian, Y. Kong, B. Zhong, and Y. Fu. Residual dense network for image super-resolution, 2018. DOI: 10.48550/ARXIV.1802.08797. URL: https://doi.org/10.48550/arxiv.1802.08797.

[13] J. Li, F. Fang, K. Mei, and G. Zhang. Multi-scale residual network for image super-resolution. In V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, editors, *Computer Vision – ECCV 2018*, pages 527–542, Cham. Springer International Publishing, 2018. ISBN: 978-3-030-01237-3. DOI: 10.1007/978-3-030-01237-3_32. URL: https://doi.org/10.1007/978-3-030-01237-3_32.

[14] N. Ahn, B. Kang, and K.-A. Sohn. Fast, accurate, and lightweight super-resolution with cascading residual network, 2018. DOI: 10.48550/ARXIV.1803.08664. URL: https://doi.org/10.48550/arxiv.1803.08664.

[15] X. Chu, B. Zhang, R. Xu, and H. Ma. Multi-objective reinforced evolution in mobile neural architecture search, 2019. DOI: 10.48550/ARXIV.1901.01074. URL: https://doi.org/10.48550/arxiv.1901.01074.

[16] X. Chu, B. Zhang, H. Ma, R. Xu, and Q. Li. Fast, accurate and lightweight super-resolution with neural architecture search, 2019. DOI: 10.48550/ARXIV.1901.07261. URL: https://doi.org/10.48550/arxiv.1901.07261.

[17] X. Luo, Y. Xie, Y. Zhang, Y. Qu, C. Li, and Y. Fu. Latticenet: towards lightweight image super-resolution with lattice block. In A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, editors, *Computer Vision – ECCV 2020*, pages 272–289, Cham. Springer International Publishing, 2020. ISBN: 978-3-030-58542-6. DOI: 0.1007/978-3-030-58542-6_17. URL: https://doi.org/0.1007/978-3-030-58542-6_17.

[18] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon. Cbam: convolutional block attention module, 2018. DOI: 10.48550/ARXIV.1807.06521. URL: https://doi.org/10.48550/arxiv.1807.06521.

[19] Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong, and Y. Fu. Image super-resolution using very deep residual channel attention networks, 2018. DOI: 10.48550/ARXIV.1807.02758. URL: https://doi.org/10.48550/arxiv.1807.02758.

[20] T. Dai, J. Cai, Y. Zhang, S.-T. Xia, and L. Zhang. Second-order attention network for single image super-resolution. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11057–11066, 2019. DOI: 10.1109/CVPR.2019.01132. URL: https://doi.org/10.1109/CVPR.2019.01132.

[21] Y. Hu, J. Li, Y. Huang, and X. Gao. Channel-wise and spatial feature modulation network for single image super-resolution, 2018. DOI: 10.48550/ARXIV.1809.11130. URL: https://doi.org/10.48550/arxiv.1809.11130.

[22] J. Liu, W. Zhang, Y. Tang, J. Tang, and G. Wu. Residual feature aggregation network for image super-resolution. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2356–2365, 2020. DOI: 10.1109/CVPR42600.2020.00243. URL: https://doi.org/10.1109/CVPR42600.2020.00243.

[23] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: inverted residuals and linear bottlenecks, 2018. DOI: 10.48550/ARXIV.1801.04381. URL: https://doi.org/10.48550/arxiv.1801.04381.

[24] R. Timofte, E. Agustsson, L. Van Gool, M.-H. Yang, L. Zhang, B. Lim, et al. Ntire 2017 challenge on single image super-resolution: methods and results. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[25] M. Bevilacqua, A. Roumy, C. Guillemot, and M.-l. A. Morel. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. In *Proceedings of the British Machine Vision Conference*, pages 135.1–135.10. BMVA Press, 2012. ISBN: 1-901725-46-4. DOI: http://dx.doi.org/10.5244/C.26.135.

[26] R. Zeyde, M. Elad, and M. Protter. On single image scale-up using sparse-representations. In J.-D. Boissonnat, P. Chenin, A. Cohen, C. Gout, T. Lyche, M.-L. Mazure, and L. Schumaker, editors, *Curves and Surfaces*, pages 711–730, Berlin, Heidelberg. Springer Berlin Heidelberg, 2012. ISBN: 978-3-642-27413-8.

[27] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 2, 416–423 vol.2, 2001. DOI: 10.1109/ICCV.2001.937655.

[28] Github project repo. URL: https://github.com/pierclgr/MPRNet-SR.

[29] Wandb project page. URL: https://wandb.ai/pierclgr/ML4CV_project.