# Memory Recursive Network for Single Image Super-Resolution

Jie Liu
State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing , China
jieliu@smail.nju.edu.cn

Minqiang Zou
State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing , China
MG1733103@smail.nju.edu.cn

Jie Tang*
State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing , China
tangjie@nju.edu.cn

Gangshan Wu
State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing , China
gswu@nju.edu.cn

## ABSTRACT

Recently, extensive works based on convolutional neural network (CNN) have shown great success in single image super-resolution (SISR). In order to improve the SISR performance while reducing the number of model parameters, some methods adopt multiple recursive layers to enhance the intermediate features. However, in the recursive process, these methods only use the output features of current stage as the input of the next stage and neglect the output features of historical stages, which degrades the performance of the recursive blocks. The long-term dependencies can only be learned implicitly during the recursive processes. To address these issues, we propose the memory recursive network (MRNet) to make full use of the output features at each stage. The proposed MRNet utilizes a memory recursive module (MRM) to generate features for each recursive stage, and then these features are fused by our proposed ShuffleConv block. Specifically, MRM adopts a memory updater block to explicitly model the long-term dependencies between the output features of historical recursive stages. The output features from the memory updater will be used as the input of the next recursive stage and will be continuously updated during the recursions. To reduce the number of parameters and ease the training difficulty, we introduce a ShuffleConv module to fuse the features from different recursive stages, which is much more effective than using plain convolutional combinations. Comprehensive experiments demonstrate that the proposed MRNet achieves state-of-the-art SISR performance while using much fewer parameters.

## CCS CONCEPTS

• **Computing methodologies** → **Computational photography**; **Reconstruction**; **Image processing**.

*Corresponding author


*MM '20, October 12–16, 2020, Seattle, WA, USA*

ACM ISBN 978-1-4503-7988-5/20/10…$15.00
https://doi.org/10.1145/3394171.3413696

## KEYWORDS

image super-resolution;recursive network;convolutional neural network

**ACM Reference Format:**
Jie Liu, Minqiang Zou, Jie Tang, and Gangshan Wu. 2020. Memory Recursive Network for Single Image Super-Resolution. In *Proceedings of the 28th ACM International Conference on Multimedia (MM '20), October 12–16, 2020, Seattle, WA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3394171.3413696

## 1 INTRODUCTION

Single image super-resolution (SISR) is the process of reconstructing high-resolution (HR) image by using a single low-resolution (LR) image, which is a typical ill-posed problem since an LR image can be downsampled from multiple HR images. Various methods have been proposed to solve this problem, such as interpolation-based methods [18, 32], reconstruction-based methods [34] and learning-based methods [4, 11, 14, 24–26, 29, 30]. In recent years, with the progress of deep learning (DL) research and the improvement of computer performance, the training process of deep neural network (DNN) is no longer an obstacle for SR tasks, so DL-based methods have become a research hotspot.

Since convolutional neural network (CNN) can make full use of the local spatial coherence of the image, using CNN to tackle the SISR task is the most commonly used method in DL-based methods [4, 14, 26]. Dong *et al.* [4] firstly proposed a three-layer CNN model named SRCNN to establish the mapping relationship between LR and HR, which has better reconstruction results compared with the conventional learning-based methods. A deeper network has larger receptive fields and can retain more contextual information, so adopting deeper networks can often achieve better performance, but it is difficult to train a deep CNN due to the vanishing gradient and exploding gradient problems. To alleviate the impact of these problems on training, residual learning [7] and dense connection [10] strategies are widely adopted by SR models [14, 21, 31, 36, 37]. Zhang *et al.* [36] introduced a residual in residual (RIR) structure to form a very deep network RCAN (400 convolutional layers), which has excellent performance with 16M parameters. Other very deep networks also have large number of parameters, such as EDSR [21] (43M) and RDN [37] (22M), these

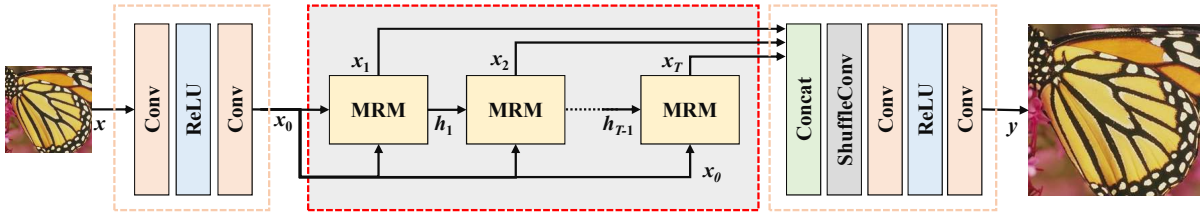



**Figure 1: The architecture of proposed MRNet.**

large-capacity networks usually consume huge computing and storage resources which are less applicable to resource-constrained equipment. At the same time, increasing the depth of the model will also increase the risk of model overfitting.

In order to obtain a larger receptive field and higher-level feature representation without introducing too many parameters, recursive learning is employed in SR task [2, 6, 15, 20, 27, 28]. A recursive network usually applies the same recursive module to process features multiple times. Kim *et al.* [15] used the same single convolutional layer in their 16-recursion DRCN, thus DRCN has the receptive field of 41 by 41, which is larger compared to SRCNN. Although DRCN can achieve superior results using a three-layer CNN, it is very difficult to learn the mapping of different levels of features with a single weight layer [15]. Tai *et al.* [27] adopted residual learning and multi-path structure to address the shortcoming of DRCN, but the output features of each recursion are not well exploited. To handle the issue of restricted long-term memory, Tai *et al.* [28] proposed MemNet which employs a recursive unit and a gate unit to explicitly mine persistent memory. However, the recursive unit in Memnet only receives the information from the last recursion. Li *et al.* [20] used a $1 \times 1$ convolution in their proposed SRFBN to refine low-level representations with high-level information, but SRFBN only adopted the features of the last recursion in each recursion to refine the output features, which can not make full use of the information generated by the entire recursive process.

To handle these drawbacks, we propose the memory recursive network (MRNet) for lightweight and accurate SISR (Figure 1). MRNet directly uses the LR image as input to reduce the amount of calculation, after extracting the shallow feature using two convolutional layers, our proposed memory recursive module (MRM) is used to enhance the features during the recursive phase. MRM adopts a memory updater to explicitly model the long-term dependencies between the output features of historical recursive stages. The output features from the memory updater will be used as the input of the next recursive stage and will be continuously updated during the recursions. In order to make full use of the features generated at different recursive stages, these features will be fused by our proposed ShuffleConv module and then produce the final HR image. $1 \times 1$ convolution is the most commonly used feature fusion method in many previous studies[2, 15, 28, 37], but when the number of input feature channels is large, it will lead to heavy parameters and optimization difficulty. We propose the ShuffleConv module to reduce the number of parameters by using the local feature fusion strategy.

In summary, Our contributions can be summarized as follows:

- We propose a recursive model named memory recursive network (MRNet) for the SISR task. MRNet makes full use of the output features at each recursive stage, thus it can achieve higher performance with fewer parameters.
- We propose an efficient recursive module called the memory recursive module (MRM), which leverages the historical features to compensate for the input feature. The historical features will also be adjusted automatically with the recursive process, so it can continuously improve the quality of input features.
- We propose a ShuffleConv module to fuse the features from different recursive stages more effectively. ShuffleConv uses local feature fusion so that it has fewer parameters than the normal $1 \times 1$ convolution.

## 2 RELATED WORK

### 2.1 Deep learning based SISR

Recently, deep-learning-based methods have shown excellent performance in the SISR task. Dong *et al.* [4] firstly proposed a three-layers CNN network named SRCNN to model the mapping between interpolated LR and HR. Kim *et al.* [14] proposed a 20-layers CNN model that employs residual learning and gradient clipping strategies to overcome the difficulty of optimization. These methods usually interpolate the LR image to the desired size before sending it to the networks, which not only introduces noise and blurring but also increases computation cost. Thus some methods performed the upsampling operation at the end of the SR model [5, 26]. Dong *et al.* [5] adopted the deconvolution layer at the end of their FSRCNN model, so most of the calculations are performed in low-dimensional space, which greatly reduces the amount of calculation and storage consumption. A sub-pixel convolution was proposed in ESPCN [26] to upsample features end-to-end, this sub-pixel layer is widely adopted by SR models [2, 21, 36, 37]. To reduce the training difficulty for large scaling factor SR model, LapSRN [16] used a cascade of CNNs to progressively reconstruct HR images, and LapSRN supervised the reconstructed image at each upsampling stage.

In terms of network design, because residual learning [7] and dense connections [10] can alleviate the vanishing gradient and exploding gradient problems, these structures are wildly used in SR tasks. SRResnet [17] used a 16-blocks deep ResNet [7] to handle the training difficult in the SR problem. Lim *et al.* [21] discarded the batch normalization layer in SRResnet and designed a very deep residual network to generate higher performance. Zhang et

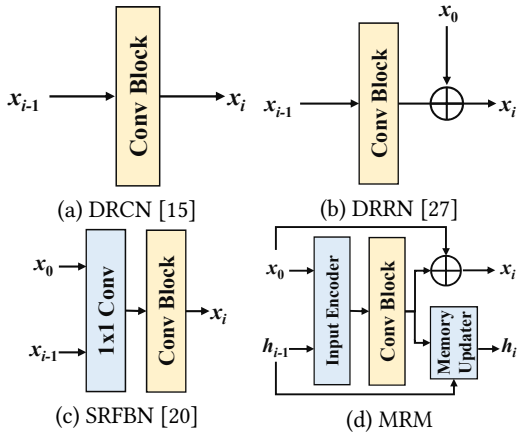



**Figure 2: Some typical structures of the recursive model and the proposed MRM.**

al.[36] adopted channel attention[9] into the residual block to form a very deep network (more than 400 convolutional layers). Tong *et al.* [31] introduced SRDensenet which uses a dense connection to strengthen feature propagation and boost feature reuse. Zhang et al.[37] combined the residual learning and dense connection in their proposed RDN to make full use of all the hierarchical features from the original LR image. In order to control the amount of parameters and calculation of the SR model, Ahn *et al.* [2] proposed an accurate and lightweight deep network using the cascading mechanism to extract features. Hui *et al.* [13] designed an information distillation network for better extracting features.

## 2.2 Recursive convolutional network for SISR

To achieve larger receptive fields and obtain high-level features without increasing overwhelming parameters, recursive learning was widely adopted in SR models [2, 6, 15, 20, 27, 28]. Usually, the recursive model will apply the same module multiple times to enhance the features in the recursive process. Kim *et al.* [15] built a 16-recursion DRCN using a single convolutional layer as the recursive unit, recursive supervision and skip connections are employed in DRCN to ease the training difficulty. Tai *et al.* [27] adopted global and local residual learning in their proposed DRRN to mitigate the difficulty of training. MemNet [28] introduced a recursive unit and a gate unit to compose the memory block, so the recursive unit can learn multi-level representations under different receptive fields. Ahn *et al.* [2] developed CARN-M which utilizes recursive cascading block to reduce the parameters. Han *et al.* [6] proposed a dual-state recurrent network (DSRN), DSRN exploits both LR and HR signals to get more accurate reconstruction features. Li *et al.* [20] proposed a feedback network (SRFBN) to refine low-level features with high-level representations, SRFBN uses a recursive module to achieve such feedback manner.

Since most of the recursive networks for SR task are similar, we summarize several typical structures in this part. As shown in Figure 2(a), we unfold the entire recursive process into $T$ iterations, for a specific iteration $i$, the output features $x_i$ in DRCN [15] can be obtained by:

$$x_i = f_{CB}(x_{i-1}) \tag{1}$$

$$x_i = f_{CB}(f_{CB}(\cdots(f_{CB}(x_0)))) \tag{2}$$

where Equation 2 is the unfolded form of Equation 1, $x_0$ denotes the initial input of the recursive module and $x_i$ is the output of the recursive module at the $i$th iteration, $f_{CB}$ denotes the operations of the convolution block. It is worth noting that MemNet [28] also adopts this type of recursive formula and uses a 1×1 convolution to fuse the output feature of each recursion.

DRRN[27] employed residual learning to ease the training difficulty, the basic operation in the recursive module at the $i$-th iteration can be formulated as:

$$x_i = x_0 + f_{CB}(x_{i-1}) \tag{3}$$

the only difference between DRCN and DRRN in the recursive module is that DRRN introduces residual learning, which allows the recursive module to learn the residual of $x_0$.

SRFBN [20] adopted a feedback block to generate high-level representations. The initial low-level input $x_0$ is enhanced by the high-level output information $x_{i-1}$ obtained at the $(i-1)$-th iteration. This operation can be described as follows:

$$x_i = f_{CB}(f_{conv1}([x_0, x_{i-1}])) \tag{4}$$

where $f_{conv1}$ denotes the 1×1 convolution. So SRFBN can use the high-level features generated by the last recursion to continuously enhance the initial input during the recursive process.

# 3 MEMORY RECURSIVE NETWORK

In this section, we will introduce the details of our proposed MRNet. Specifically, we start with the overall architecture of MRNet, then describe the memory updating mechanism in our proposed MRM about how it leverages the historical features to generate input features for the next iteration. Finally, we explain the detailed calculation process of our proposed ShuffleConv and the differences between ShuffleConv and the plain convolutions.

## 3.1 Network architecture

As shown in Figure 1, the proposed MRNet mainly consists of three parts: shallow feature extraction block (SFE), recursive feature enhance block (RFE) and a reconstruction block (RB). Let's denote $x$ and $y$ as the input LR image and output HR image of MRNet, respectively. The output shallow feature $x_0$ can be obtained by:

$$x_0 = f_{SFE}(x) \tag{5}$$

where $f_{SFE}$ represents the function of shallow feature extraction block, we use two convolution layers and one ReLU layer in SFE. Then the low-level feature $x_0$ is sent to the recursive feature enhance block to learn high-level features.

In order to better elaborate the calculation process of RFE, we unfold RFE along the temporal direction to a fixed $T$ iterations. As shown in the red box in Figure 1, the MRM in the figure are shared between iterations. The detailed information about MRM will be introduced in Section 3.2. In this part, we simplify MRM to consist of two inputs and two outputs. The operation of MRM at the $i$-th iteration can be formulated as:

$$x_i, h_i = f_{MRM}(x_0, h_{i-1}) \tag{6}$$

where $x_i$ is the output features generated by MRM at the $i$-th iteration, $h_i$ denotes the historical memory information at the $i$-th iteration, $f_{MRM}$ is the operation collection in MRM. It is worth noting that in the first recursion ($i$=1), we use $x_0$ as the historical memory information ($h_0 = x_0$).

After $T$ iterations, MRM will generate $T$ features $(x_1, x_2, \cdots, x_T)$, these features are then fed into the reconstruction block to generate the HR image. The operation in RB can be described as:

$$y = f_{RC}(f_{SC}([x_1, x_2, \cdots, x_T])) \tag{7}$$

where $f_{RC}$ denotes the final reconstruction convolutions in Figure 1, which has two convolution layer and one ReLU layer. $f_{SC}$ is the operations in ShuffleConv, the details about ShuffleConv will be shown in Section 3.3. Since features generated at different iterations contain abundant information for reconstruction, these hierarchical features are aggregated by ShuffleConv for a better SR performance.

## 3.2 Memory recursive module

From the summary in Section2.2, we note that most recursive models only use the output features of current stage as the input of the next stage and neglect the output features of historical stages, which hinders the performance of the model. The long-term dependencies can only be learned implicitly during the recursive processes. To address these problems, we propose a memory recursive module (MRM), which consists of an input encoder (IE) and a memory updater (MU). As shown in Figure 2(d), the memory updater is responsible for fusing historical features. The operations in MRM at the $i$-th iteration can be formulated as:

$$m_i = f_{IE}([x_0, h_{i-1}]) \tag{8}$$

$$r_i = f_{CB}(m_i) \tag{9}$$

$$x_i = r_i + x_0 \tag{10}$$

$$h_i = f_{MU}([r_i, h_{i-1}]) \tag{11}$$

where $f_{IE}$ is the operations in the input encoder, $m_i$ denotes the encoded memory information which will be used as the input of the following convolution block, $f_{CB}$ denotes the function of the convolution block, $r_i$ is the output of the convolution block. $f_{MU}$ represents the function of the memory updater. Both input encoder and memory updater are implemented using the 1×1 convolution to keep the model lightweight enough for SISR.

MRM first uses the input encoder to encode the initial low-level feature $x_0$ and historical information $h_{i-1}$ to obtain the fused memory feature $m_i$. After extracting features using the convolution block, the output feature $r_i$ is then used to update the historical information (Equation 11) and obtain the output feature (Equation 10) at the $i$-th iteration. We employ residual learning in Equation 10 to get a better convergence. The proposed MRM can enhance the input feature more effectively and make the input of the convolution block more stable. Since the historical information $h_i$ is updated according to $r_i$ and $h_{i-1}$, the output feature $r_i$ is fully utilized by subsequent iterations.

As for the convolution block $f_{CB}$ in MRM, we adopt the same residual block as EDSR [21]. As shown in Figure 3, the convolution block consists of multiple residual blocks and has a convolution layer at the end. Residual learning enables the module to extract features more effectively.

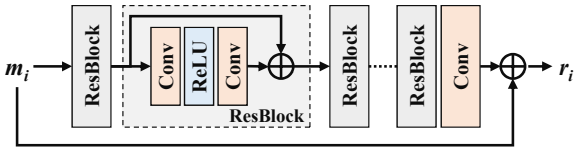



**Figure 3: The structure of ConvBlock used in MRM.**

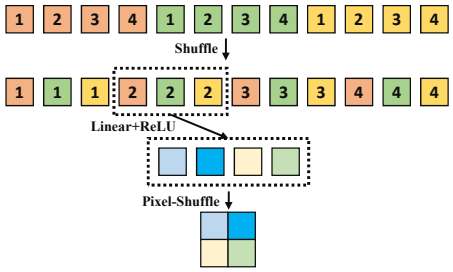



**Figure 4: The architecture of proposed ShuffleConv.**

## 3.3 ShuffleConv module

To make full use of the hierarchical features produced by MRM, we propose the ShuffleConv module that can aggregate features more effectively. The conventional way for feature fusion is to employ a $1 \times 1$ convolution [2, 15, 28, 37]. However, when the volume of input features is very large and redundant, it is not efficient enough to use the $1 \times 1$ convolution for feature fusion. In the scenario of recursive learning, the output features generated by different iterations are strongly correlated since the convolution weights are shared between iterations. Besides, when the number of recursion is very large, the $1 \times 1$ convolution would introduce a large number of parameters. Let's denote $x_{ij}$ as the feature map of the $j$-th channel for the $i$-th iteration. Because $x_{aj}$ and $x_{bj}$ are obtained by the same set of convolution weights, the correlation between them is stronger than others. Here, we propose the ShuffleConv module to perform local feature fusion by grouping the strongly correlated features, which can aggregate features more effectively and save a considerable number of parameters.

As shown in Figure 4, for simplicity, we suppose that there has only one pixel in the feature map, the feature channel is 4, the scaling factor is 2 and the total number of iterations is 3. Squares of the same color in the figure belong to the same feature maps in a iteration and the numbers represent the channel indexes of the feature maps. We first group these features according to their channel indexes. For features in each group, a linear layer and ReLU function are adopted to generate a specific number of channels. Finally, Pixel-Shuffle[26] is utilized to get the HR feature map.

Let $c_0$ represent the number of channels for output features in MRM, the total number of iterations is $T$, and the scaling factor is $s$. For conventional 1×1 convolution, the number of input channels is $T \times c_0$, and the number of output channel before the Pixel-Shuffle layer is $c_0 \times s^2$, so the total number of parameters is $T \times c_0 \times c_0 \times s^2$. In our proposed ShuffleConv, the parameter of each linear layer is $T \times s^2$, and the total number of linear layer is $c_0$, thus the total

parameter in ShuffleConv is $T \times c_0 \times s^2$. So the parameter of 1×1 convolution is $c_0$ times that of ShuffleConv.

It should be noted that ShuffleNet [35] and our proposed ShuffleConv have some similar ideas. The purpose of shuffle operation in ShuffleNet is to enable cross-group information interchange because the group convolution layer hinders the information flow. In ShuffleConv, we use the shuffle operation to group feature maps of strong correlation to fuse the features locally and then generate the high-resolution feature maps.

### 3.4 Discussions

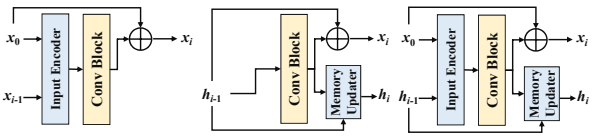
**Difference to MemNet.** MemNet [28] is designed to achieve persistent memory. The memory block in MemNet is mainly composed of a recursive unit and a gate unit, the recursive unit can generate multi-level representations in the recursive process and these representations will be sent to the gate unit to control the long-term and short-term memory. The differences between MRNet and MemNet are: 1) MemNet only adopts the output features in the last iteration as the input of the recursive unit in their proposed memory block, which neglects the difference between these features and not fully utilizes the output features in each recursion. In our proposed MRNet, we use an input encoder to fuse the initial feature and historical feature. The output features in each recursion are utilized to update historical information and sent to the ShuffleConv module to get the reconstructed HR features, so these multi-level features can be fully utilized in MRNet. 2) MemNet adopts a gate unit to learn the weights for different memories adaptively which is more similar to dense connection, and too many input memories make it impossible to effectively use these memories due to training difficulty. Our proposed MRNet updates memories by the last memories and the current output, which has fewer parameters and is easier to optimize, and these memories can be utilized more efficiently in the next recursion.

**Difference to SRFBN.** SRRFBN [20] employs a feedback mechanism to refine low-level representations with high-level information. The feedback mechanism is implemented by supervising each recursive process. The main differences between MRNet and SRFBN are: 1) The feature extraction block used in SRFBN is a downsample-upsample unit, while our proposed MRNet only adopts the most commonly used residual structure to prove the effectiveness of MRM. 2) SRFBN employs a feedback mechanism to refine low-level features, in our proposed MRNet, we only use common feedforward methods to continuously use historical information to enhance the original input features. 3) SRFBN refines the initial input only using the high-level feature in the last iteration, but the historical information utilized by MRNet will be updated at each iteration with the last historical information and the current output, which can refine the initial input more effectively.

## 4 EXPERIMENTS

### 4.1 Settings

In our proposed MRNet, the kernel size in the first convolution layer of SFE is 7×7 to obtain enough information from the original LR image. The number of channels of all feature maps in the intermediate layers is set to 64 ($c_0$=64). The ConvBlock in MRM uses 10 ResBlocks to extract features. We adopt different iterations for different scaling factors to ensure performance while keeping the model lightweight enough. The number of iterations is 4, 6 and 8 when the scaling factor is 2, 3 and 4 respectively.




(a) Input encoder (b) Memory updater (c) MRM

**Figure 5: Modules used in ablation study.**

**Table 1: Ablation results of different recursive modules used in MRNet. The scaling factor is ×2, we use PSNR (dB) values after training 400 epochs.**

| Method | Params | Set5 | Set14 |
|---|---|---|---|
| Baseline model | 862K | 37.84 | 33.46 |
| Residual model | 862K | 37.89 | 33.49 |
| Input encoder | 870K | 37.92 | 33.51 |
| Memory updater | 870K | 37.90 | 33.54 |
| MRM | 878K | 37.94 | 33.63 |

Following [2, 20, 21, 36, 37], we train the proposed MRNet with 800 training images of DIV2K dataset [1]. During training, for each batch of input, we randomly select 16 LR RGB patches with the size 48×48, then data augmentation is performed by flipping horizontally or vertically or rotating 90°. We set 1000 iterations of back-propagation as an epoch and use ADAM optimizer with $\beta_1 = 0.9$, and $\epsilon = 10^{-8}$. The learning rate is initialized as $10^{-4}$ and decreases to half for every $2{\times}10^5$ iterations during the total $10^6$ iterations. The loss function we used is $L1$ loss. We use PyTorch framework to implement the proposed MRNet with a Nvidia 1080Ti GPU. For evaluation, we adopt five standard benchmark datasets: Set5 [3], Set14 [33], BSD100 [22], Urban100 [12] and Manga109 [23]. We evaluate the results using two commonly used evaluation metrics: peak signal-to-noise ratio (PSNR) and structural similarity (SSIM). The SR results are calculated on Y channel of transformed YCbCr space. As in the previous works [2, 4, 15, 21, 27, 28, 36, 37], we adopt the Bicubic degradation model in this paper.

### 4.2 Effects of memory recursive module

*4.2.1 Ablation study.* To investigate the effect of the input encoder and memory updater, we set up a set of ablation experiments. As shown in Figure 5, memory updater is removed in Figure 5(a), and because there is no historical information $h_{i-1}$ in Figure 5(a), the output $x_{i-1}$ of last iteration is used as the input. It is worth noting that this structure is similar to the structure of SRFBN (Figure 2(c)). We then remove the input encoder in MRM and use $h_{i-1}$ as the input of the module (Figure 5(b)). To better illustrate the effectiveness of MRM, we also train a baseline model using the same recursive structure as DRCN (Figure 2(a)) and a residual model like DRRN (Figure 2(b)). We set the scaling factor to ×2 and train all models for 400 epochs to ensure the stability of the results.

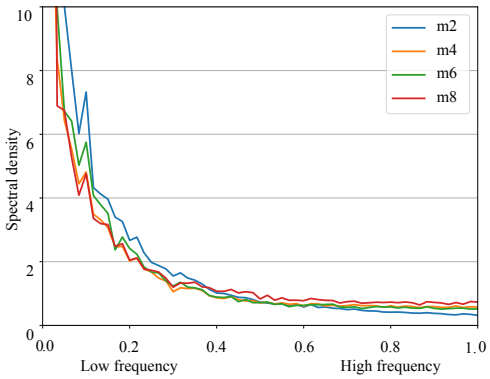



**Figure 6: Spectral densities of the average feature maps of $m_i$ for different recursions.**

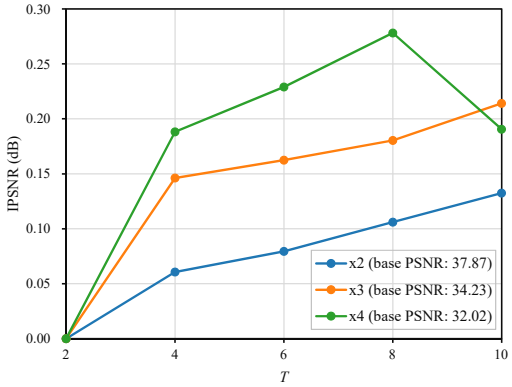



**Figure 7: IPSNR on Set5 under different recursions. The base PSNR in the label represents the PSNR result when the recursion number is 2.**

**Table 2: Performance comparison of the ShuffleConv and baseline methods at different recursions. The scaling factor is ×2, we use PSNR (dB) values after training 400 epochs.**

| Recursion | | 2 | 4 | 6 | 8 | 10 |
|---|---|---|---|---|---|---|
| baseline | Params | 1033K | 1041K | 1049K | 1058K | 1066K |
| | Set5 (dB) | 37.87 | 37.93 | 37.97 | 38.00 | 38.02 |
| ShuffleConv | Params | 882K | 887K | 891K | 896K | 900K |
| | Set5 (dB) | 37.88 | 37.99 | 38.02 | 37.97 | 38.00 |

As shown in Table 1. The baseline model adopts the recursive structure used in DRCN, where the output features of each iteration are not fully utilized, so the reconstruction performance is not very satisfactory. Adopting residual learning like DRRN improves the results slightly, but the historical information is not well exploited. Using the input encoder can increase the performance since it can fuse the initial input $x_0$ and the output feature $x_{i-1}$ of the last iteration, which makes the input features of ConvBlock more stable. By utilizing the memory updater to encode historical information, the results are also better than the baseline model. Comparing the performance of baseline model and MRM, we find that MRM can improve 0.17dB on Set14 without increasing too many parameters, which proves the effectiveness of the proposed MRM. We can draw conclusions from the above analysis that the proposed MRM can utilize the features generated by different recursions more effectively than previous recursive modules [15, 20, 27].

*4.2.2 Frequency domain analysis of MRM.* To further analyze the changes of features in different recursions, we analyze the encoded features $m_i$ of MRM in the frequency domain. Inspired by [20, 28], we first perform a fast Fourier transform on the average feature maps of $m_i$ and center their spectrum. Then we estimate the mean of spectral densities for continuous frequency ranges by placing concentric circles. The visualization results are depicted in Figure 6. By analyzing the spectral density of $m_i$, we can conclude that the output features of input encoder are continuously enhanced during the recursive process so that the low frequency parts are suppressed and the high frequency parts are strengthened, which further proves the effectiveness of the MRM module.

## 4.3 Effects of ShuffleConv

Our proposed MRNet uses ShuffleConv in the reconstruction block to aggregate features from different recursions. To validate the effectiveness of ShuffleConv, we compare it with the conventional $1 \times 1$ convolution under different number of recursions. Specifically, we set the number of recursions ($T$) to 2, 4, 6, 8, and 10, and the scaling factor is ×2. For comparison, we use 1×1 convolution for feature fusion and 3×3 convolution for HR feature generation in the baseline model. In order to ensure that the network could be fully trained, we train 400 epochs for both models. Table 2 shows the detailed performance comparison. As we can see, the ShuffleConv method can achieve similar results with the baseline while using much fewer parameters, which indicates the effectiveness of our method.

## 4.4 Study of recursion

Recursion depth is the main factor affecting the performance of the recursive model. In this section, we study the effect of increasing recursion depth with different scaling factors. Firstly, we set the number of recursions ($T$) to 2, 4, 6, 8, and 10. Then we train the proposed MRNet by setting the scaling factors to ×2, ×3, and ×4. All 15 models are trained with 400 Epochs to ensure the stability of the results. The results are depicted in Figure 7. To better visualize the results, at each scaling factor, we calculate the PSNR difference between different recursion with $T = 2$ (IPSNR). As we can see, most of the time our proposed MRNet can be continuously improved when increasing $T$. Meanwhile, we notice that when the scaling factor is ×4, the performance of $T = 10$ is worse than $T = 8$, and we also find that the training loss at $T = 10$ is greater than $T = 8$ during the entire training process. We speculate that the main reason for its performance degradation is that when the scaling factor is large, the LR input is more rough in spatial content so it just need a modest number of recursions to make the network focus on current spatial context.

**Table 3: Average PSNR/SSIM for scaling factor ×2, ×3 and ×4. Best and second-best results are highlighted in red/blue text.**

| Method | Scale | Params | MAC | Set5 | Set14 | BSD100 | Urban100 | Manga109 |
|---|---|---|---|---|---|---|---|---|
| | | | | PSNR/SSIM | PSNR/SSIM | PSNR/SSIM | PSNR/SSIM | PSNR/SSIM |
| Bicubic | ×2 | - | - | 33.66/0.9299 | 30.24/0.8688 | 29.56/0.8431 | 26.88/0.8403 | 30.80/0.9339 |
| VDSR[14] | | 0.67M | 612.6G | 37.53/0.9587 | 33.03/0.9124 | 31.90/0.8960 | 30.76/0.9140 | 37.22/0.9750 |
| DRCN[15] | | 1.77M | 17974.3G | 37.63/0.9588 | 33.04/0.9118 | 31.85/0.8942 | 30.75/0.9133 | 37.55/0.9732 |
| DRRN[27] | | 0.30M | 6796.9G | 37.74/0.9591 | 33.23/0.9136 | 32.05/0.8973 | 31.23/0.9188 | 37.88/0.9749 |
| IDN[13] | | 0.55M | 123.5G | 37.83/0.9600 | 33.30/0.9148 | 32.08/0.8985 | 31.27/0.9196 | 38.01/0.9749 |
| MemNet[28] | | 0.68M | 2662.4G | 37.78/0.9597 | 33.28/0.9142 | 32.08/0.8978 | 31.31/0.9195 | 37.72/0.9740 |
| CARN[2] | | 1.59M | 222.8G | 37.76/0.9590 | 33.52/0.9166 | 32.09/0.8978 | 31.92/0.9256 | 38.36/0.9765 |
| MDSR_baseline[21] | | 3.23M | 411.2G | 37.98/0.9605 | 33.58/0.9181 | 32.18/0.8997 | 32.09/0.9284 | 38.48/0.9768 |
| MSRN[19] | | 5.89M | 1356.8G | 38.08/0.9605 | 33.74/0.9170 | 32.23/0.9013 | 32.22/0.9326 | 38.82/0.9868 |
| OISR-RK2[8] | | 4.97M | 1145.7G | 38.12/0.9609 | 33.80/0.9193 | 32.26/0.9006 | 32.48/0.9317 | 38.78/0.9774 |
| MRNet-S (Ours) | | 0.88M | 776.5G | 38.06/0.9607 | 33.81/0.9197 | 32.25/0.9004 | 32.40/0.9309 | 38.78/0.9774 |
| MRNet (Ours) | | 1.06M | 776.5G | 38.14/0.9610 | 33.83/0.9199 | 32.28/0.9008 | 32.63/0.9330 | 39.00/0.9777 |
| Bicubic | ×3 | - | - | 30.39/0.8682 | 27.55/0.7742 | 27.21/0.7385 | 24.46/0.7349 | 26.95/0.8556 |
| VDSR[14] | | 0.67M | 612.6G | 33.66/0.9213 | 29.77/0.8314 | 28.82/0.7976 | 27.14/0.8279 | 32.01/0.9340 |
| DRCN[15] | | 1.77M | 17974.3G | 33.82/0.9226 | 29.76/0.8311 | 28.80/0.7963 | 27.15/0.8276 | 32.24/0.9343 |
| DRRN[27] | | 0.30M | 6796.9G | 34.03/0.9244 | 29.96/0.8349 | 28.95/0.8004 | 27.53/0.8378 | 32.71/0.9379 |
| IDN[13] | | 0.55M | 54.8G | 34.11/0.9253 | 29.99/0.8354 | 28.95/0.8013 | 27.42/0.8359 | 32.71/0.9381 |
| MemNet[28] | | 0.68M | 2662.4G | 34.09/0.9248 | 30.00/0.8350 | 28.96/0.8001 | 27.56/0.8376 | 32.51/0.9369 |
| CARN[2] | | 1.59M | 118.8G | 34.29/0.9255 | 30.29/0.8407 | 29.06/0.8034 | 28.06/0.8493 | 33.50/0.9440 |
| MDSR_baseline[21] | | 3.23M | 202.3G | 34.37/0.9270 | 30.33/0.8427 | 29.10/0.8055 | 28.16/0.8529 | 33.52/0.9444 |
| MSRN[19] | | 6.08M | 621.2G | 34.38/0.9262 | 30.34/0.8395 | 29.08/0.8041 | 28.08/0.8554 | 33.44/0.9427 |
| OISR-RK2[8] | | 5.64M | 578.6G | 34.55/0.9282 | 30.46/0.8443 | 29.18/0.8075 | 28.50/0.8597 | 33.79/0.9464 |
| MRNet-S (Ours) | | 0.88M | 526.6G | 34.52/0.9281 | 30.45/0.8445 | 29.18/0.8072 | 28.50/0.8598 | 33.88/0.9465 |
| MRNet (Ours) | | 1.06M | 526.6G | 34.68/0.9292 | 30.56/0.8463 | 29.24/0.8083 | 28.70/0.8633 | 34.19/0.9481 |
| Bicubic | ×4 | - | - | 28.42/0.8104 | 26.00/0.7027 | 25.96/0.6675 | 23.14/0.6577 | 24.89/0.7866 |
| VDSR[14] | | 0.67M | 612.6G | 31.35/0.8838 | 28.01/0.7674 | 27.29/0.7251 | 25.18/0.7524 | 28.83/0.8870 |
| DRCN[15] | | 1.77M | 17974.3G | 31.53/0.8854 | 28.02/0.7670 | 27.23/0.7233 | 25.14/0.7510 | 28.93/0.8854 |
| DRRN[27] | | 0.30M | 6796.9G | 31.68/0.8888 | 28.21/0.7720 | 27.38/0.7284 | 25.44/0.7638 | 29.45/0.8946 |
| IDN[13] | | 0.56M | 30.9G | 31.82/0.8903 | 28.25/0.7730 | 27.41/0.7297 | 25.41/0.7632 | 29.41/0.8942 |
| MemNet[28] | | 0.68M | 2662.4G | 31.74/0.8893 | 28.26/0.7723 | 27.40/0.7281 | 25.50/0.7630 | 29.42/0.8942 |
| CARN[2] | | 1.59M | 90.9G | 32.13/0.8937 | 28.60/0.7806 | 27.58/0.7349 | 26.07/0.7837 | 30.47/0.9084 |
| MDSR_baseline[21] | | 3.23M | 138.0G | 32.16/0.8947 | 28.60/0.7823 | 27.58/0.7367 | 26.07/0.7858 | 30.48/0.9080 |
| SRDenseNet[31] | | 2.02M | 389.9G | 32.02/0.8934 | 28.50/0.7782 | 27.53/0.7337 | 26.05/0.7819 | - |
| MSRN[19] | | 6.33M | 365.1G | 32.07/0.8903 | 28.60/0.7751 | 27.52/0.7273 | 26.04/0.7896 | 30.17/0.9034 |
| SRFBN [20] | | 3.63M | 7466.1G | 32.39/0.8970 | 28.77/0.7860 | 27.68/0.7400 | 26.47/0.7980 | 30.96/0.9140 |
| OISR-RK2[8] | | 5.50M | 412.2G | 32.32/0.8965 | 28.72/0.7843 | 27.66/0.7390 | 26.37/0.7953 | 30.76/0.9123 |
| MRNet-S (Ours) | | 0.89M | 403.8G | 32.39/0.8972 | 28.77/0.7854 | 27.66/0.7391 | 26.41/0.7960 | 30.90/0.9130 |
| MRNet (Ours) | | 1.06M | 403.8G | 32.51/0.8987 | 28.81/0.7868 | 27.73/0.7412 | 26.60/0.8015 | 31.14/0.9155 |

## 4.5 Comparison with the state-of-the-arts

In order to further demonstrate the performance of our proposed MRNet, we adopt the same multi-scale structure as MDSR [21] to train the final MRNet. Specifically, for networks with different scaling factors, our MRNet uses different SFEs and RBs, and the middle RFE module is shared. Therefore, under different scaling factors, MRNet first uses different SFEs to extract shallow features and then uses the same MRM module to enhance the features. Finally, features generated by the MRM are fed to the specific RB module to generate the HR images.

We compare our MRNet with several state-of-the-art methods: VDSR [14], DRCN [15], DRRN [27], IDN [13], MemNet [28], CARN [2], SRDenseNet [31], MSRN [19], SRFBN [20], and OISR-RK-2 [8]. As shown in Table 3, The MRNet-S model refers to our MRNet with single-scale structure. Multiply-accumulate operations (MAC) are calculated under the assumption that the size of the output image is 1280×720. Note that SRFBN [20] uses DIV2K+Flickr2K dataset

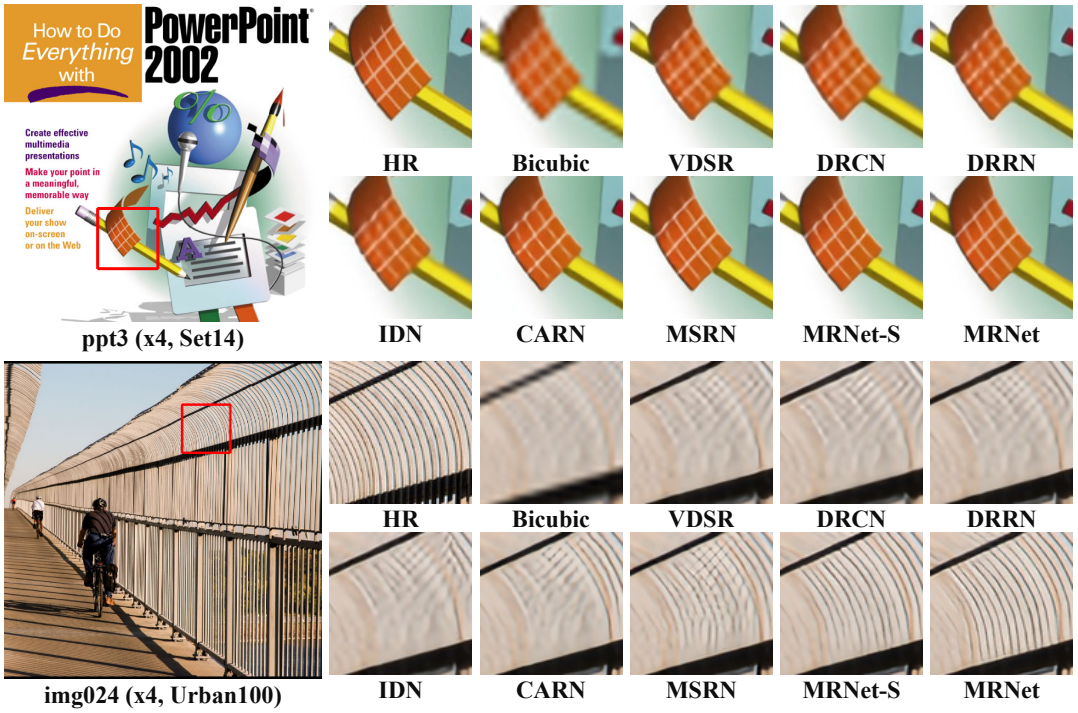



**Figure 8: Visual comparisons of MRNet with other SOTA SR methods with scaling factor ×4. Zoom in for a better visual experience.**

(DF2K, 800+2650, 2K images) to train their model, the author also provides the results of SRFBN trained using the DIV2K dataset at ×4 scaling factor, which are included in Table 3. Our proposed MRNet-S can achieve better results than most networks with fewer parameters, and the multi-scale version MRNet can take full advantage of the multi-scale training. It adopts the same MRM to enhance the features for different scaling factors, which achieves better results than MRNet-S. In comparison to the multi-scale model MDSR-baseline, which has seven times as many parameters as MRNet, MRNet can also achieve better results with much fewer parameters. Figure 9 shows the comparison of PSNR versus the number of parameters, the performance of our MRNet greatly surpasses some previous recursive models like DRCN, DRRN and MemNet. Compared with SRFBN trained under the same training data, our model can also achieve better results with fewer parameters and MAC (Table 3).

As shown in Figure 8, for "ppt3" from Set14, our MRNet can completely reconstruct the alignment of lines, while the lines predicted by other methods are discontinuous. For "ima024" from Urban100, the original input has been unable to distinguish these railings, other models have not correctly reconstructed the area, but our network has reconstructed the shape and direction of the railings.

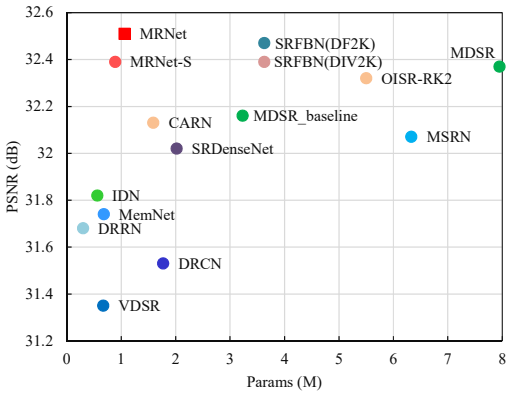



**Figure 9: Comparison of PSNR *vs.* parameters on Set5 ×4.**

## 5 CONCLUSION

In this paper, we proposed a novel recursive model for SISR called memory recursive network (MRNet) to make the most of the output feature of each recursion. By adopting the proposed memory recursive module (MRM) in each recursion, the historical information of each stage can be used more effectively. We also propose a ShuffleConv module to fuse the features from different recursion, which is more lightweight than the conventional $1 \times 1$ convolution. Comprehensive experiments show the MRNet composed of these proposed modules can better handle the features of different recursions and enables the ConvBlock in the MRM to enhance the features more accurately. At the same time, MRNet can boost the SISR performance when increasing the recursion times appropriately. The comparison on the benchmark datasets demonstrates that our MRNet could achieve better performance with fewer parameters.

## REFERENCES


[1] E. Agustsson and R. Timofte. 2017. NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*.

[2] Namhyuk Ahn, Byungkon Kang, and Kyung-Ah Sohn. 2018. Fast, Accurate, and Lightweight Super-Resolution with Cascading Residual Network. *arXiv preprint arXiv:1803.08664* (2018).

[3] Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie line Alberi Morel. 2012. Low-Complexity Single-Image Super-Resolution based on Nonnegative Neighbor Embedding. In *Proceedings of the British Machine Vision Conference*. BMVA Press, 135.1–135.10. https://doi.org/10.5244/C.26.135

[4] C. Dong, C. C. Loy, K. He, and X. Tang. 2016. Image super-resolution using deep convolutional networks. *IEEE TPAMI* 38, 2 (2016), 295–307.

[5] C. Dong, C. C. Loy, and X. Tang. 2016. Accelerating the super-resolution convolutional neural network. In *ECCV*. Springer, 391–407.

[6] Wei Han, Shiyu Chang, Ding Liu, Mo Yu, Michael Witbrock, and Thomas S Huang. 2018. Image super-resolution via dual-state recurrent networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[7] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep residual learning for image recognition. In *CVPR*. 770–778.

[8] Xiangyu He, Zitao Mo, Peisong Wang, Yang Liu, Mingyuan Yang, and Jian Cheng. 2019. ODE-inspired Network Design for Single Image Super-Resolution. In *2019 IEEE Conference on Computer Vision and Pattern Recognition*.

[9] Jie Hu, Li Shen, and Gang Sun. 2018. Squeeze-and-Excitation Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[10] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Wein-berger. 2017. Densely connected convolutional networks.. In *CVPR*.

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

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

[13] Z. Hui, X. Wang, and X. Gao. 2018. Fast and Accurate Single Image Super-Resolution via Information Distillation Network. In *CVPR*. 723–731.

[14] J. Kim, J. K. Lee, and K. M. Lee. 2016. Accurate image super-resolution using very deep convolutional networks. In *CVPR*. 1646–1654.

[15] J. Kim, J. K. Lee, and K. M. Lee. 2016. Deeply-recursive convolutional network for image super-resolution. In *CVPR*. 1637–1645.

[16] W. Lai, J. Huang, A. Narendra, and M. Yang. 2017. Deep Laplacian Pyramid Networks for Fast and Accurate Super-Resolution. In *IEEE Conferene on Computer Vision and Pattern Recognition*.

[17] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunning-ham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. 2017. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network.. In *CVPR*.

[18] Lei Zhang and Xiaolin Wu. 2006. An edge-guided image interpolation algorithm via directional filtering and data fusion. *IEEE Transactions on Image Processing* 15, 8 (Aug 2006), 2226–2238. https://doi.org/10.1109/TIP.2006.877407

[19] Juncheng Li, Faming Fang, Kangfu Mei, and Guixu Zhang. 2018. Multi-scale Residual Network for Image Super-Resolution. In *The European Conference on Computer Vision (ECCV)*.

[20] Zhen Li, Jinglei Yang, Zheng Liu, Xiaomin Yang, Gwanggil Jeon, and Wei Wu. 2019. Feedback Network for Image Super-Resolution. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[21] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee. 2017. Enhanced deep residual networks for single image super-resolution. In *CVPRW*.

[22] D. Martin, C. Fowlkes, D. Tal, and J. Malik. 2001. 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*, Vol. 2. 416–423 vol.2. https://doi.org/10.1109/ICCV.2001.937655

[23] Yusuke Matsui, Kota Ito, Azuma Fujimoto Yuji Aramaki, Toru Ogawa, Toshihiko Yamasaki, and Kiyoharu Aizawa. 2017. Sketch-based manga retrieval using manga109 dataset. *Multimedia Tools and Applications* 76 (2017), 21811âĂŞ21838.

[24] T. Peleg and M. Elad. 2014. A Statistical Prediction Model Based on Sparse Representations for Single Image Super-Resolution. *IEEE Transactions on Image Processing* 23, 6 (2014), 2569–2582.

[25] S. Schulter, C. Leistner, and H. Bischof. 2015. Fast and accurate image upscaling with super-resolution forests. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3791–3799.

[26] W. Shi, J. Caballero, F. HuszÃąr, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. 2016. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *CVPR*. 1874–1883.

[27] Y. Tai, J. Yang, and X. Liu. 2017. Image super-resolution via deep recursive residual network. In *CVPR*.

[28] Y. Tai, J. Yang, X. Liu, and C. Xu. 2017. Memnet: A persistent memory network for image restoration. In *CVPR*. 4539–4547.

[29] R. Timofte, V. De, and L. V. Gool. 2013. Anchored Neighborhood Regression for Fast Example-Based Super-Resolution. In *2013 IEEE International Conference on Computer Vision*. 1920–1927.

[30] Radu Timofte, Vincent DeÂăSmet, and Luc VanÂăGool. 2015. A+: Adjusted Anchored Neighborhood Regression for Fast Super-Resolution. In *Computer Vision – ACCV 2014*, Daniel Cremers, Ian Reid, Hideo Saito, and Ming-Hsuan Yang (Eds.). Springer International Publishing, Cham, 111–126.

[31] T. Tong, G. Li, X. Liu, and Q. Gao. 2017. Image super-resolution using dense skip connections. In *ICCV*. IEEE, 4809–4817.

[32] Xin Li and M. T. Orchard. 2001. New edge-directed interpolation. *IEEE Transactions on Image Processing* 10, 10 (Oct 2001), 1521–1527. https://doi.org/10.1109/83.951537

[33] Roman Zeyde, Michael Elad, and Matan Protter. 2012. On Single Image Scale-Up Using Sparse-Representations. In *Curves and Surfaces*, Jean-Daniel Boissonnat, Patrick Chenin, Albert Cohen, Christian Gout, Tom Lyche, Marie-Laurence Mazure, and Larry Schumaker (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 711–730.

[34] K. Zhang, X. Gao, D. Tao, and X. Li. 2012. Single Image Super-Resolution With Non-Local Means and Steering Kernel Regression. *IEEE Transactions on Image Processing* 21, 11 (2012), 4544–4556.

[35] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 6848–6856.

[36] Y. Zhang, K. Li, K. Li, L. Wang, B. Zhang, and Y. Fu. 2018. Image super-resolution using very deep residual channel attention networks. *ECCV* (2018).

[37] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. 2018. Residual Dense Network for Image Super-Resolution. In *CVPR*.