Practical Single-Image Super-Resolution Using Look-Up Table

Younghyun Jo Seon Joo Kim
Yonsei University

Abstract

*A number of super-resolution (SR) algorithms from interpolation to deep neural networks (DNN) have emerged to restore or create missing details of the input low-resolution image. As mobile devices and display hardware develops, the demand for practical SR technology has increased. Current state-of-the-art SR methods are based on DNNs for better quality. However, they are feasible when executed by using a parallel computing module (*e.g. *GPUs), and have been difficult to apply to general uses such as end-user software, smartphones, and televisions. To this end, we propose an efficient and practical approach for the SR by adopting look-up table (LUT). We train a deep SR network with a small receptive field and transfer the output values of the learned deep model to the LUT. At test time, we retrieve the precomputed HR output values from the LUT for query LR input pixels. The proposed method can be performed very quickly because it does not require a large number of floating point operations. Experimental results show the efficiency and the effectiveness of our method. Especially, our method runs faster while showing better quality compared to bicubic interpolation.*

1. Introduction

The goal of single-image super-resolution (SR) is to generate high-resolution (HR) results with sufficient high-frequency details from the corresponding low-resolution (LR) input image. Interpolation based methods were dominant early on, where the missing pixel values were estimated by the weighted average of the nearby pixels with known values. Some examples of the interpolation based approach include bilinear, bicubic [19], and Lanczos. The methods are intuitive and fast, however, they hardly restore missing details as the same interpolation weights are applied regardless of the image structure. As a result, the results of interpolation based SR look overly blurry.

To create HR images with better quality, diverse approaches have been proposed. Example based methods exploit a database of LR-HR image patch pairs generated from a number of external training images [3, 11, 10],

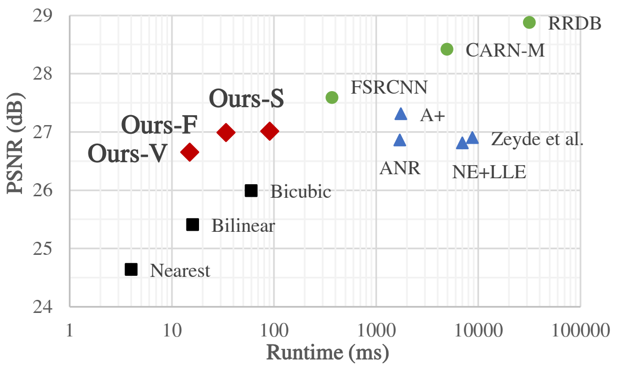

Figure 1. PSNR comparison on Set14 testset for $\times 4$ SR and runtime is measured for generating 1280×720 output image on a Samsung Galaxy S7 smartphone (except for sparse coding based methods). We compare our method with several interpolation based methods (square), sparse coding based methods (triangle), and deep learning based methods (circle). Our methods show faster or comparable runtime compared to bicubic interpolation while achieving much better PSNR quality.

or exploit self-similarity from a test image itself [12, 47]. One downside is that searching for the nearest neighbor patch is time consuming [5]. Sparse coding based methods [48, 50, 42, 43], which learn a compact representation of the patches, have also been popular and showed promising results. However, computing the sparse representation of the input patch requires a high computational cost.

As deep learning showed powerfulness in various computer vision tasks, attempts to use deep neural networks (DNN) for SR exploded [8, 20, 9, 22, 26, 2, 13, 53, 52, 44, 25, 36, 31, 33]. They achieve state-of-the-art SR performance in terms of peak signal-to-noise ratio (PSNR), however, it comes with a number of multiplication operations from numerous convolutional layers. Therefore, a special parallel computing device such as graphic processing units (GPU) or tensor processing units (TPU) is essential to handle high computational complexity and memory consumption. The DNN based solutions are difficult to apply in practice without the special hardware, and this is one of the major obstacles for practical SR.

Despite the emergence of a variety of approaches for SR, the interpolation based methods are still commonly used as the base algorithm in image processing software such as Photoshop, Matlab, and OpenCV, because they are simple and practical. While much effort has been made to improve the visual quality of SR, relatively little consideration has been paid for the feasibility of practical SR to be applied in end-user software and hardware like consumer cameras, surveillance cameras, mobile phones, and televisions which have a limited number of calculation units. Recently, running deep models on mobile devices is becoming practical thanks to GPU attached mobile processors and many efforts in optimizations [16]. Still, as another line of study, it is important to develop practical methods for more general situations when GPUs are not available.

To this end, we propose a practical single-image SR method which runs faster or comparably compared to bicubic interpolation while achieving much better quality. We employ a look-up table (LUT) approach, which is commonly used in embedded systems to accelerate computation. For a complicated function or a series of computations, if we compute output values once and put them in a LUT, then all we need to do afterward is to just retrieve the values without performing the computation again. Therefore, LUTs are effective when the computation time is much longer than the memory access time. In image processing, LUTs have long been used in a variety of color transfer tasks and within the standard profiles of the International Color Consortium because of their excellence [34]. In addition, LUTs have been used in camera imaging pipeline due to the hardware friendly property [17]. Similarly, LUTs can accelerate the overall runtime of an SR algorithm by just retrieving the precomputed outputs from the memory, because a series of floating point operations is much slower than memory access.

In this paper, we train a deep SR network under certain constraints to map output values of the learned deep model to a LUT. We constrain the receptive field (RF) of a deep SR network to be small (up to 4 pixels). With the small RF, we can compute all possible output values of the learned network as the output value is determined only depending on a small number of input values. After the training, the output values are saved into the LUT (up to 4D LUT, same as RF size). We name the built LUT as SR-LUT. In practice, our SR-LUTs have uniformly sampled points of values. Therefore, at test time, we first look-up the precomputed values of the nearby sampling points and they are properly merged by an interpolation technique for the final output. In particular, for the efficiency in 4D LUT, we extend the tetrahedral interpolation in 3D space to 4D. In Fig. 1, we compare our method with several SR methods. Compared to bicubic interpolation, our very fast and fast models with RF size 2 and 3 (Ours-V and Ours-F respectively) run faster and our slow model with RF size 4 (Ours-S) runs in comparable time, and all of our models achieve better PSNR value. Also, our methods have comparable PSNR values with faster speed than sparse coding based methods (triangle). Although deep learning based methods (circle) obviously achieve better accuracy, they have difficulty in achieving faster runtime beyond a certain level, restricting their practical use.

In summary, the contributions of this paper are:

- We introduce a simple and novel method for fast and practical single-image SR by transferring input and output values from a learned deep SR model to a LUT (SR-LUT). To the best of our knowledge, this is the first time to demonstrate the benefits of LUTs for single-image SR.
- Our method is inherently faster because we just retrieve the precomputed values from the LUT on memory, instead of executing heavy computations composed of a large number of floating point multiplication and addition operations as in the previous SR methods. We verify the efficiency of our method in experiments on a smartphone, and our fast model runs faster than bicubic interpolation.
- Our method can be easily implemented on both software and hardware as a memory array, without a special computing module such as GPUs. We believe that this merit will enable our approach to be used in diverse practical SR applications.

2. Related Work

2.1. Fast Super-Resolution

As sparse coding based methods showed good SR performance [48, 50, 4], several methods for running those approaches faster have been introduced. In ANR [42] and A+ [43], the authors computed a projection matrix, which maps LR input feature to the HR output patch, in advance from a learned sparse dictionary. At test time, the HR output is obtained by using the precomputed projection matrix. This approach improved the speed by 5-10 times from the baseline method [50]. Specifically, ANR takes more than a second for generating 1280×720 output image from 320×180 input image on our desktop computer (Intel Xeon CPU E3-1230 v3 @ 3.30GHz with 32GB RAM).

RAISR [37] learned a set of filters for the SR task with low computational complexity and achieved faster runtime than the previous works. However, the method is not fast as the interpolation because the method computes image gradient and singular value decomposition for each patch of the input image for hash table keys. In addition, there are also other types of fast SR methods [35, 38, 41].

While there were efforts for fast SR, the methods still take a long time compared to bicubic interpolation, which is executed in almost constant time. In contrast, our LUT based approach is inherently faster than the above methods

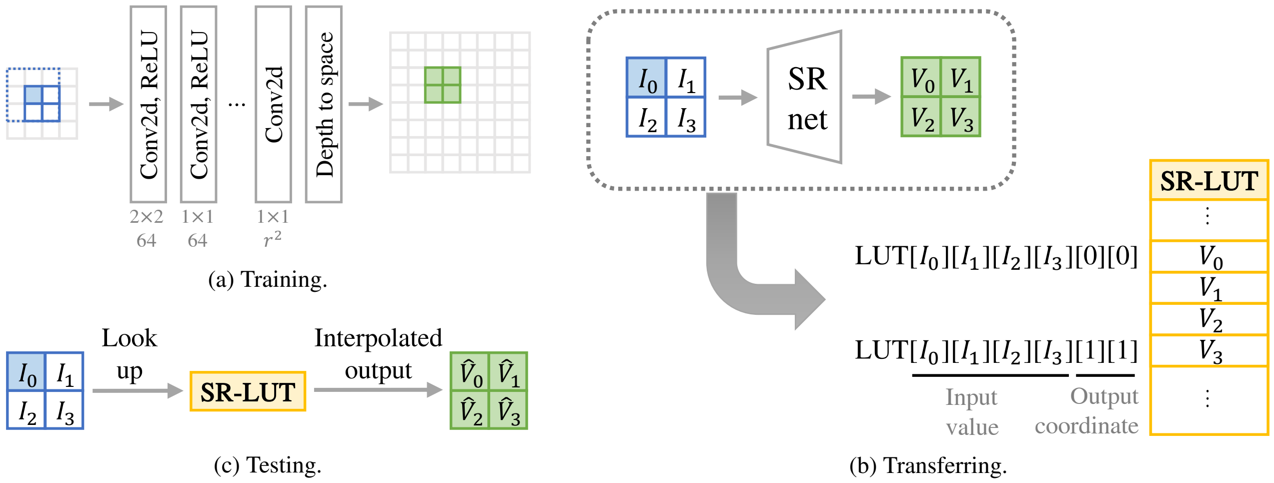

Figure 2. The overview of our method. The figure is depicted for $\times 2$ SR ($r = 2$) and RF size 4 with 2×2 convolutional kernel (Ours-S). (a) A small deep SR network is trained with the limited RF size, to transfer the outputs to a LUT after the training. 2×2 kernel (solid line) covers 3×3 area (dotted line) by rotational ensemble (Sec. 3.1), and green colored 4 pixels are the output of blue colored input pixel. (b) The output values of the learned network are saved to 4D SR-LUT at the location indexed by the corresponding 4 input values. (c) At test time, SR is performed solely using the SR-LUT.

as it only requires very little computation for generating the output. As shown in Fig. 1, our methods run faster or are as fast as bicubic interpoloation on a mobile device. Note that researches combining DNN and LUT for efficiency have been emerging, such as in photo enhancement [49].

2.2. Real-Time Deep Super-Resolution

Real-time SR algorithms based on deep learning with GPUs have also been proposed. In early DNN based SR works, faster runtime is achieved by using a small number of convolutional layers. ESPCN [39] and FSRCNN [9] used 3 and 8 convolutional layers respectively with small feature dimension size. However, it is difficult to achieve state-of-the-art performance by using a small model capacity.

Following researches have focused on reducing large models while minimizing the performance degradation [15, 51, 28]. CARN-M [2] reduced the number of parameters and multiplication-addition operations of their original big model CARN by replacing the conventional convolutions with group convolutions. PAMS [24] quantized original full precision 32bit SR models to 8 or 4bit for the model compression. BSRN [46] further quantized the original model to binary precision. In addition, FALSR [7], ESRN [40] and TPSR [23] employ neural architecture search algorithms to find fast and accurate lightweight deep SR networks from given efficient convolutional building blocks.

As we have seen, there have been a variety of efforts toward practical SR. Some of the methods are executed in real-time on CPUs, however, none has targeted to run on limited computing resources such as mobile devices. As they consist of convolutional layers that require a good number of floating point multiplication and addition operations, the runtime will increase in devices with limited computing resources. In practice, FSRCNN takes 77ms on our desktop and 371ms on a Samsung Galaxy S7 smartphone. In contrast, our proposed SR-LUTs do not require such computational overhead, and our fast model takes 34ms on the smartphone without too much degradation in performance. Our method can be an appropriate choice for devices with low computational capability where the deep methods are difficult to be applied.

3. Method

The overview of our method is shown in Fig. 2 To obtain an SR-LUT, we first train a lightweight deep SR network with a small RF size (Fig. 2a), and transfer the output values of the deep model to the SR-LUT (Fig. 2b). At test time, for an input LR patch, corresponding HR pixel values are obtained from the SR-LUT (Fig. 2c).

For a practical SR-LUT, the RF size of the deep model should be small because the size of SR-LUT increases exponentially as the RF size increases. Note that, it is very difficult to exploit existing deep models to LUT as the range of possible input and output values is too large. In Table 1, we estimate the size of SR-LUTs for upscaling factor r when storing 8bit input and output values. For example, when the RF size is 2 and $r = 4$, the full LUT size is calculated as $(2^8)^2 \times 4^2 \times$ 8bit $=$ 1MB because $(2^8)^2$ LUT entries are needed (2^8 bins for 8bit input value) and each entry has 4^2 number of 8bit output values. Similarly, the full SR-LUTs take 256MB, 64GB, and 16TB when the RF size is 3, 4, and

RF	LUT	Full size	Sampled size
1 pixel	1D	4KB	272B
2 pixels	2D	1MB	4.516KB
3 pixels	3D	256MB	76.766KB
4 pixels	4D	64GB	1.274MB
5 pixels	5D	16TB	21.665MB
n pixels	nD	$(2^8)^n \times r^2$B	$(2^4+1)^n \times r^2$B

Table 1. SR-LUT size estimation when storing 8bit output value for 8bit input image with upscaling factor $r = 4$. Full LUT has 2^8 entries for each input pixel and sampled LUT size is calculated for 2^4 entries. Sampled LUT is necessary when RF size is greater than or equal to 3 for practicality.

5 pixels respectively. Because the full SR-LUT size is too large when the RF size is greater than or equal to 3, we use a sampled LUT in practice. We empirically found that the RF size should be less than or equal to 4 pixels for practical implementation because it affects the runtime. We set the RF size as 2, 3, and 4 for Ours-V, Ours-F, and Ours-S respectively. Ours-V and Ours-F are faster than Ours-S, but the visual quality of Ours-S is better. The following contents in this section will be explained based on RF size 4 with 2×2 convolutional kernel (Ours-S configuration), but other RF sizes are also applicable. Due to the limited RF size, each color channel has to be processed independently.

3.1. Training Deep SR Network

Network Architecture Because the network is constrained by a very small RF size, a large number of convolution layers does not increase the accuracy boundlessly but converges faster. Therefore, we make use of a deep network consisting of 6 convolutional layers followed by ReLU activation except for the last layer (Fig. 2a). For the RF size 4, the kernel size of the first layer is set to 2×2 and that of the rest layers is set to 1×1. A different kernel shape (*e.g.* 1×4) can be used for the first layer. However, 2×2 shape is the most appropriate as it takes into account the most 4 relevant adjacent input pixels, and it shows better visual quality in experiments (Sec. 4.3). The number of features in the convolutional layers is set to 64 and that of the last layer is set to r^2. The output blob of the network is reshaped to the desired size through the depth to space operation [39]. We note that the number of layers does not affect the final runtime as the deep model is only used for building the corresponding SR-LUT.

Rotational Ensemble Training In general, the performance of the SR task can be improved when more pixels are considered. However, our RF size 4 (2×2) is too small for accurate HR image estimation. For example, the RF size of FSRCNN [9] is 169 pixels (13×13) and even bicubic interpolation exploits 16 (4×4) nearest neighbor pixels. In order to exploit more area in LR input, we use a rotational ensemble in the training phase. For our deep network, 4 rotational ensemble with 0, 90, 180, and 270 degrees covers total 9 (3×3) LR pixels (blue dotted area shown in Fig. 2a with respect to blue colored reference pixel). Each output from the 4 rotations is summed up for generating the final output. Formally, the final output $\hat{y}_i$ can be expressed as follows:

$$\hat{y}_i = \frac{1}{4}\sum_{j=0}^{3} R_j^{-1}\Big(f\big(R_j(x_i)\big)\Big), \tag{1}$$

where x_i is LR input patch, f is the deep SR network, R_j is image rotation operation to $j \times 90$ degree, and R_j^{-1} is the reverse rotation operation. The SR network f is trained by using a pixel reconstruction loss:

$$\sum_i \ell(\hat{y}_i, y_i), \tag{2}$$

where y_i is GT target patch and ℓ is the mean squared error.

The rotational self-ensemble strategy has been used to maximize the accuracy only at test time in previous deep SR works [26, 52]. We apply this strategy further at the training time to improve the performance while having a small RF size. This is very effective for our directional kernel shape, which helps to achieve good results. Even the RF is too small compared to other SR methods, we can effectively consider more area for accurate SR without increasing the LUT size through this strategy.

3.2. Transferring to LUT

After training the deep SR network, we build a 4D SR-LUT for the RF size 4 (Fig. 2b). Note that Fig. 2b shows a 6D LUT but we call it 4D LUT by only considering the input value dimensions. For the full LUT, we compute the output values of the learned deep network for all possible input values and save them to the LUT. The input value is used as the index of the LUT, and the corresponding output value is stored at that location. In practice, we use a uniformly sampled LUT as the size of the full LUT is very large (64GB). Specifically, we uniformly divide the original input space of 2^8 bins (0-255 for 8bit input image) into 2^4+1 bins. In other words, we sample the points by equally spacing the original input space with the sampling interval size of 2^4. As a result, the uniformly sampled 4D SR-LUT has the output values at input points of 0, 16, ..., 240, and 255 (the last point) for all dimensions, and the size is reduced to 1.274MB (4D LUT[256][256][256][256] is reduced to LUT[17][17][17][17]). At test time, the values of nonsampled points are interpolated by using the values of the nearest sampled points. We have tested the sampling interval range from 2^2 to 2^8 and verified the original performance is almost maintained until the interval size of 2^4 in experiments (Sec. 4.3).

LUT	Interpolation	Multiplications	Comparisons
2D	Bilinear	4	0
	Triangular	3	1
3D	Trilinear	11	0
	Tetrahedral	4	2.5
4D	Tetralinear	26	0
	4-simplex	5	4.5

Table 2. The number of multiplication and comparison operations of linear interpolation and tetrahedral interpolation equivalents for multidimensional LUTs. We use the latter approach as it takes overall faster runtime in practice.

For better accuracy, nonuniform sampling techniques have been often used in color transfer tasks [27, 32]. However, the overall runtime increases as they require extra computation and comparison to locate the index of the nearest sampled points. This contrasts with our goal of a fast and practical SR approach. Therefore, we use the uniform sampling for ease of implementation and faster runtime.

3.3. Testing Using SR-LUT

Once the SR-LUT is built, SR is performed solely with the SR-LUT (Fig. 2c). In case of using a full LUT, the output HR value is directly retrieved from the LUT. On the other hand, in case of using a sampled LUT, an appropriate interpolation technique is required to generate the output value by using the values of the nearest sampled points. To index the nearest sampled points, we simply take most significant bits (MSB) of the input pixel value as we use equally sampled LUT. This can be just done by masking and shifting the bits. For the 8bit input pixel value, we take 4 MSBs then it locates one of the nearest sampled points.

To interpolate the values of the found nearest sampled points, one can use linear interpolation as a baseline method. Namely, bilinear, triliear, tetralinear, and pentalinear interpolation for 2D, 3D, 4D, and 5D SR-LUTs respectively. Instead, we use well known tetrahedral interpolation approach [18] as it is faster than trilinear interpolation for 3D LUT. Tetrahedral interpolation calculates the weighted sum of the values of bounding 4 vertices of a tetrahedron in 3D space. In Table 2, we compare the number of operations of linear interpolation and tetrahedral interpolation equivalents for multidimensional LUTs. For 3D LUT, trilinear interpolation needs at least 11 multiplications while tetrahedral interpolation only needs 4 multiplications with 2.5 comparison operations (*if-else*). The overall runtime of tetrahedral interpolation including the comparison operations is faster than trilinear interpolation in practice [18], and it is also same to other dimensions.

For easy understanding of the 4D equivalent of tetrahedral interpolation, we explain the 2D equivalent triangular interpolation in Fig. 3. For query input $I_0 = 24$ ($00011000_{(2)}$) and $I_1 = 60$ ($00111100_{(2)}$), we first split the input values by 4 MSBs and 4 least significant bits (LSB). The values of MSBs, 1 and 3 for I_0 and I_1 respectively, are used for determining the nearest sampled points. The values of LSBs, $L_x = 8$ and $L_y = 12$ for I_0 and I_1 respectively, are used for determining the bounding triangle and the weights of the bounding vertices. Two bounding vertices are fixed at $P_{00} = \text{LUT}[1][3]$ and $P_{11} = \text{LUT}[1+1][3+1]$, and the other vertex is determined by comparing L_x and

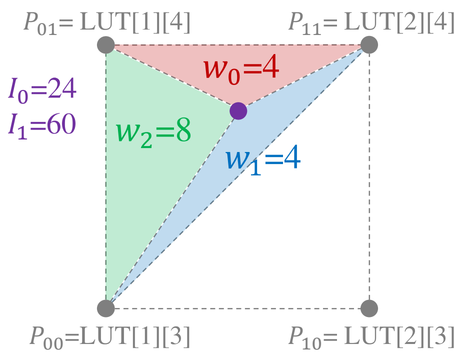

Figure 3. An example of triangular interpolation for a sampled 2D LUT with the sampling interval of 2^4. For query input $I_0 = 24$ and $I_1 = 60$, the nearest points P_{00}, P_{01}, and P_{11}, and the corresponding weights w_0, w_1, and w_2 are determined. The output value is calculated as the weighted sum. The same principle applies to 3D and 4D LUTs.

Condition	w_0	w_1	w_2	w_3	w_4	O_1	O_2	O_3
$L_x>L_y>L_z>L_t$	$W-L_x$	L_x-L_y	L_y-L_z	L_z-L_t	L_t	P_{1000}	P_{1100}	P_{1110}
$L_x>L_y>L_t>L_z$	$W-L_x$	L_x-L_y	L_y-L_t	L_t-L_z	L_z	P_{1000}	P_{1100}	P_{1101}
$L_x>L_t>L_y>L_z$	$W-L_x$	L_x-L_t	L_t-L_y	L_y-L_z	L_z	P_{1000}	P_{1001}	P_{1101}
$L_t>L_x>L_y>L_z$	$W-L_t$	L_t-L_x	L_x-L_y	L_y-L_z	L_z	P_{0001}	P_{1001}	P_{1101}
$L_x>L_z>L_y>L_t$	$W-L_x$	L_x-L_z	L_z-L_y	L_y-L_t	L_t	P_{1000}	P_{1010}	P_{1110}
$L_x>L_z>L_t>L_y$	$W-L_x$	L_x-L_z	L_z-L_t	L_t-L_y	L_y	P_{1000}	P_{1010}	P_{1011}
$L_x>L_t>L_z>L_y$	$W-L_x$	L_x-L_t	L_t-L_z	L_z-L_y	L_y	P_{1000}	P_{1001}	P_{1011}
$L_t>L_x>L_z>L_y$	$W-L_t$	L_t-L_x	L_x-L_z	L_z-L_y	L_y	P_{0001}	P_{1001}	P_{1011}
$L_z>L_x>L_y>L_t$	$W-L_z$	L_z-L_x	L_x-L_y	L_y-L_t	L_t	P_{0010}	P_{1010}	P_{1110}
$L_z>L_x>L_t>L_y$	$W-L_z$	L_z-L_x	L_x-L_t	L_t-L_y	L_y	P_{0010}	P_{1010}	P_{1011}
$L_z>L_t>L_x>L_y$	$W-L_z$	L_z-L_t	L_t-L_x	L_x-L_y	L_y	P_{0010}	P_{0011}	P_{1011}
$L_t>L_z>L_x>L_y$	$W-L_t$	L_t-L_z	L_z-L_x	L_x-L_y	L_y	P_{0001}	P_{0011}	P_{1011}
$L_y>L_x>L_z>L_t$	$W-L_y$	L_y-L_x	L_x-L_z	L_z-L_t	L_t	P_{0100}	P_{1100}	P_{1110}
$L_y>L_x>L_t>L_z$	$W-L_y$	L_y-L_x	L_x-L_t	L_t-L_z	L_z	P_{0100}	P_{1100}	P_{1101}
$L_y>L_t>L_x>L_z$	$W-L_y$	L_y-L_t	L_t-L_x	L_x-L_z	L_z	P_{0100}	P_{0101}	P_{1101}
$L_t>L_y>L_x>L_z$	$W-L_t$	L_t-L_y	L_y-L_x	L_x-L_z	L_z	P_{0001}	P_{0101}	P_{1101}
$L_y>L_z>L_x>L_t$	$W-L_y$	L_y-L_z	L_z-L_x	L_x-L_t	L_t	P_{0100}	P_{0110}	P_{1110}
$L_y>L_z>L_t>L_x$	$W-L_y$	L_y-L_z	L_z-L_t	L_t-L_x	L_x	P_{0100}	P_{0110}	P_{0111}
$L_y>L_t>L_z>L_x$	$W-L_y$	L_y-L_t	L_t-L_z	L_z-L_x	L_x	P_{0100}	P_{0101}	P_{0111}
$L_t>L_y>L_z>L_x$	$W-L_t$	L_t-L_y	L_y-L_z	L_z-L_x	L_x	P_{0001}	P_{0101}	P_{0111}
$L_z>L_y>L_x>L_t$	$W-L_z$	L_z-L_y	L_y-L_x	L_x-L_t	L_t	P_{0010}	P_{0110}	P_{1110}
$L_z>L_y>L_t>L_x$	$W-L_z$	L_z-L_y	L_y-L_t	L_t-L_x	L_x	P_{0010}	P_{0110}	P_{0111}
$L_z>L_t>L_y>L_x$	$W-L_z$	L_z-L_t	L_t-L_y	L_y-L_x	L_x	P_{0010}	P_{0011}	P_{0111}
else	$W-L_t$	L_t-L_z	L_z-L_y	L_y-L_x	L_x	P_{0001}	P_{0011}	P_{0111}

Table 3. Tetrahedral interpolation equivalent for 4D space (interpolation inside a 4-simplex). There are total 24 cases depending on the values of the LSBs of input values.

L_y. In this example, $P_{01} = \text{LUT}[1][3+1]$ is selected because $L_x < L_y$, otherwise, P_{10} is selected (We omit the output value coordinate for simplicity). The weight of each vertex is the area of the opposite triangle and calculated as $w_0 = W - L_y$, $w_1 = L_y - L_x$, and $w_2 = L_x$, where $W = 2^4$ (the sampling interval). Finally, the output value is calculated as weighted sum as follows: $\hat{V} = (w_0 P_{00} + w_1 P_{01} + w_2 P_{02})/W$.

Likewise, tetrahedral interpolation can be extended to 4D space by using the values of bounding 5 vertices of 4-simplex geometry. One 4-simplex is selected among total 24 cases depending on the values of the LSBs (L_x, L_y, L_z, L_t for I_0, I_1, I_2, and I_3 respectively). In Table 3, we show the weights w_i and the bounding vertices O_i for each case. The output value is then calculated as weighted sum as follows:

$$\hat{V} = \frac{1}{W} \sum_{i=0}^{4} w_i O_i, \tag{3}$$

where $O_0 = P_{0000}$ and $O_4 = P_{1111}$.

We can also apply the same approach for 5D LUT. Just 6 multiplications are needed for interpolation within 5-simplex geometry. However, the total case increases to 120 and we empirically found this affects the runtime. Therefore, we set the maximum RF size as 4.

4. Experiments

4.1. Experimental Setting

For training, we use DIV2K [1] dataset which has been widely used in deep SR methods. The DIV2K dataset contains 800 training images with 2K resolution and covers diverse contents from cityscapes to natural sceneries. In experiments, we fix the upscaling factor to 4 (*i.e.* $r = 4$). Our SR deep model is trained for 2×10^5 iterations using Adam optimizer [21] with learning rate of 10^{-4} and mini-batch size of 32. After the training, we transfer the output values of the learned deep model to the SR-LUT.[1]

For testing, we use 5 common testsets which have been widely used in single-image SR task evaluation – Set5, Set14, BSDS100 [29], Urban100 [14] and Manga109 [30]. For quantitative evaluation, we use PSNR and structural similarity index (SSIM) [45], which are traditionally used for image quality assessment. In addition, we measure the runtime of each method on a Samsung Galaxy S7 smartphone to verify the feasibility for real application.

4.2. Comparison with Others

We compare our method with various single-image SR methods from interpolation to DNN based methods. We choose 3 interpolation based methods – nearest neighbor, bilinear, and bicubic interpolation, 4 sparse coding based methods – NE + LLE [6], Zeyde *et al.* [50], ANR [42], and A+ [43], and 3 DNN based methods – FSRCNN [9], CARN-M [2], and RRDB [44]. Note that all the sparse coding based methods used the same dictionary learned in [50]. FSRCNN is one of the fastest deep single-image SR models, CARN-M is one that balances the speed and the accuracy, and RRDB is one that shows the best accuracy.

The quantitative results are shown in Table 4. Ours-V, Ours-F, and Ours-S use 2D full LUT, 3D sampled LUT, and 4D sampled LUT respectively. The runtime is measured for generating 1280×720 output RGB image from 320×180 input, and is measured 10 times and then averaged. We use Pytorch for implementing the DNN based methods on the smartphone. However, it is difficult to implement the sparse coding based methods on the smartphone. The runtimes of the sparse coding based methods are measured by using Matlab on the desktop computer with Intel Xeon E3-1230 v3 CPU and 32GB RAM, and it may be worse if executed on the smartphone. One key merit of our method is that LUT can be implemented without any special framework such as Matlab or Pytorch, and it makes our method easy to implement in software and hardware.

Compared to bicubic interpolation, Ours-V achieves much faster runtime (-45ms) and better PSNR and SSIM values with a good margin (+0.8dB and +0.0203 respectively for Set5 testset). Similary, Ours-F also runs faster (-26ms) and shows a considerable performance gap (+1.35dB and +0.0328 for Set5). Ours-S takes a little more runtime (+31ms), but shows the best visual quality among our models. Compared to the sparse coding based methods, our results show better PSNR and SSIM performance while having smaller size LUTs than their dictionary, except for A+. A+ shows better PSNR and SSIM values than Ours-S (+0.45dB and +0.0124 for Set5), however, Ours-S runs faster and takes about 12 times smaller memory space. The DNN based methods show state-of-the-art PSNR and SSIM, but require a longer runtime. FSRCNN shows a performance gap compared to our method (+0.89dB and +0.0178 better than Ours-S for Set5), but shows 4, 11, and 25 times slower runtime compared to Ours-S, Ours-F, and Ours-V respectively. To sum up, we verify the faster runtime with the moderate accuracy for practical usage, as our method has much less computation overhead in theory.

Visual comparisons are shown in Fig. 4. We show 3 images of natural texture, text, and artificial structure. For the first two rows, Ours-V and Ours-F results show some artifacts due to limited RF sizes. For the last row, Ours-F result looks partly better than Ours-S. This is because the kernel shapes of Ours-V and Ours-F (1×2 and 1×3 respectively) can consider more pixels horizontally and vertically, whereas the kernel of Ours-S (2×2) can consider pixels diagonally. Compared to bicubic interpolation, our results generally look sharper. In some examples, Ours-S shows

[1]Code is available at https://github.com/yhjo09/SR-LUT.

	Method	Runtime	Size	Set5		Set14		BSDS100		Urban100		Manga109	
				PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
Interpolation	Nearest	4ms	-	26.25	0.7372	24.65	0.6529	25.03	0.6293	22.17	0.6154	23.45	0.7414
	Bilinear	16ms	-	27.55	0.7884	25.42	0.6792	25.54	0.6460	22.69	0.6346	24.21	0.7666
	Bicubic	60ms	-	28.42	0.8101	26.00	0.7023	25.96	0.6672	23.14	0.6574	24.91	0.7871
SR-LUT	Ours-V	**15ms**	1MB	29.22	0.8304	26.65	0.7258	26.33	0.6880	23.68	0.6852	26.30	0.8246
	Ours-F	34ms	**77KB**	29.77	0.8429	26.99	**0.7372**	**26.57**	**0.6990**	23.94	0.6971	**26.87**	0.8367
	Ours-S	91ms	1.274MB	**29.82**	**0.8478**	**27.01**	0.7355	26.53	0.6953	**24.02**	**0.6990**	26.80	**0.8380**
Sparse coding	NE + LLE [6]	7016ms*	1.434MB	29.62	0.8404	26.82	0.7346	26.49	0.6970	23.84	0.6942	26.10	0.8195
	Zeyde *et al.* [50]	8797ms*	1.434MB	26.69	0.8429	26.90	0.7354	26.53	0.6968	23.90	0.6962	26.24	0.8241
	ANR [42]	1715ms*	1.434MB	29.70	0.8422	26.86	0.7368	26.52	0.6992	23.89	0.6964	26.18	0.8214
	A+ [43]	1748ms*	15.171MB	30.27	0.8602	27.30	0.7498	26.73	0.7088	24.33	0.7189	26.91	0.8480
DNN	FSRCNN [9]	371ms	12K†	30.71	0.8656	27.60	0.7543	26.96	0.7129	24.61	0.7263	27.91	0.8587
	CARN-M [2]	4955ms	412K†	31.82	0.8898	28.29	0.7747	27.42	0.7305	25.62	0.7694	29.85	0.8993
	RRDB [44]	31717ms	16698K†	32.68	0.8999	28.88	0.7891	27.82	0.7444	27.02	0.8146	31.57	0.9185

Table 4. Quantitative comparisons on 5 common single-image SR testsets for $r = 4$. Best values are shown in bold among our models. Ours-V and Ours-F are faster than bicubic interpolation and achieve better PSNR and SSIM values with a good margin. Ours-S is slower but shows better visual quality. Runtime is measured on a Samsung Galaxy S7 smartphone for generating 1280×720 output image. *: Runtime is measured on the desktop. †: The number of parameters of DNN.

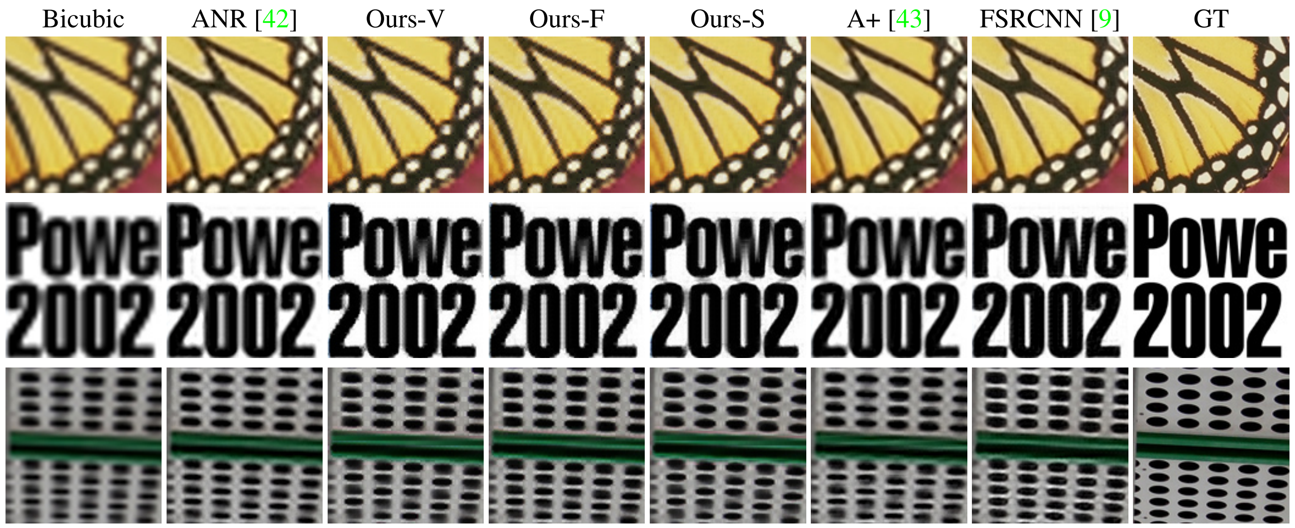

Figure 4. Qualitative results. All of our results show improved sharpness compared to bicubic interpolation. Ours-S shows the best visual quality among our models, by having smooth edge especially for diagonal direction.

improved clarity than A+ (the second row) and shows a similar level of sharpness to FSRCNN (the first row). Please see supplementary for more results.

4.3. Analysis

RF Size and Kernel Shape Experiments are conducted on various configurations of our method to analyze the effect of RF size. We test for the RF size from 2 to 4, and the detailed configurations and the results for Set5 testset are shown in Table 5. The RF size of 2 with 1×2 kernel (Ours-V) exploits total 5 input pixels by using the rotational ensemble. Similarly, the RF size of 3 with 1×3 kernel (Ours-F) covers 9, and the RF size of 4 with 1×4 kernel (configuration B) covers 13 input pixels. Larger RF sizes result in better image quality, but require longer runtime due to the higher computational complexity for interpolating output values. Expanding from 2D to 3D SR-LUT increases the runtime by 19ms, and expanding from 3D to 4D SR-LUT increases the runtime by 57ms. Likewise, the runtime for 5D SR-LUT will be much larger, making practical use difficult. For practical implementation, the tradeoff between accuracy and speed should be considered.

We also experiment on different kernel shapes. Configuration A and Ours-F have the same RF size of 3 with 1×3

	Configuration	Runtime	Size	PSNR	SSIM
Ours-V	1×2 / 4 / 5	**15ms**	1MB	29.22	0.8304
A	1×3 / 2 / 5	22ms	**77KB**	27.75	0.7892
Ours-F	1×3 / 4 / 9	34ms	**77KB**	29.77	0.8429
B	1×4 / 4 / 13	91ms	1.274MB	**29.88**	0.8463
Ours-S	2×2 / 4 / 9	91ms	1.274MB	29.82	**0.8478**
Ours-S w/o RE	2×2 / 1 / 4	29ms	1.274MB	28.32	0.8118

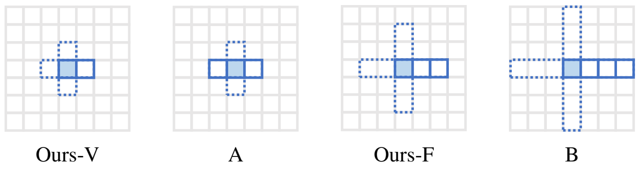

Table 5. Experiments on different RF sizes and kernel shapes. The numbers in configuration mean kernel shape, the number of rotations, and total covering pixels, in order. We also depict the kernel shape (solid line) and covering area (dotted line) by using the rotational ensemble.

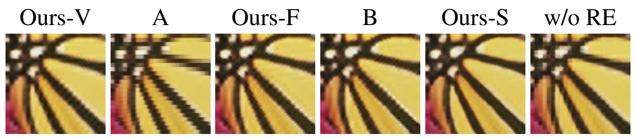

Figure 5. Visual comparison for different RF sizes and kernel shapes. Ours-S shows the best visual quality.

kernel, but the A has the symmetric kernel shape with 2 rotations. Ours-F shows much better PSNR and SSIM values than the A, and we infer this is because the asymmetric kernel needs to only focus on single direction information at a time. This seems to more noticeable in our very small RF size setting. Configuration B and Ours-S has the same RF size of 4, however, the kernel shape of B and Ours-S is 1×4 and 2×2 respectively. They show almost equivalent performance in quantitative comparison. However, Ours-S shows more visually pleasing results as shown in Fig. 5 as 2×2 kernel considers the most 4 relevant adjacent input pixels. In addition, the performance degrades if the rotational ensemble is not applied (Ours-S w/o RE).

Sampling Interval Here, we conduct the experiments on the sampling interval sizes (bin sizes) of the SR-LUT based on Ours-S model. Increasing the sampling size reduces the size of LUT, however, the original accuracy is damaged because nonsampled points should be interpolated from the sampled points. The results for the sampling size from 2^2 to 2^8 on Set5 testset are summarized in Table 6. For sampling size 2^3, the size of LUT decreases from 64GB to 18MB while maintaining the original PSNR and SSIM values. For sampling size 2^4, the size of LUT greatly decreases to 1.274MB while minimizing the loss of the origianl performance (-0.08dB and -0.0026 for PSNR and SSIM respectively). Therefore, we choose the size 2^4 as our default setting for Ours-V, Ours-F, and Ours-S. If the LUT size matters, sampling size 2^5 and 2^6 could be a better option. From the sampling size 2^6 to 2^8, we should pay attention to use because noticeable artifacts appear as shown in Fig. 6. For practical implementation, again, the tradeoff between the accuracy and the size should be considered.

Sampling	Size	PSNR	SSIM
2^0 (Full LUT)	64GB	29.90	0.8504
2^2	272MB	29.90	0.8505
2^3	18MB	29.89	0.8500
2^4 (Ours-S)	1.274MB	29.82	0.8478
2^5	102KB	29.62	0.8419
2^6	9.891KB	29.18	0.8293
2^7	1.392KB	28.50	0.8097
2^8	384B	26.64	0.7495

Table 6. Comparisons of different sampling interval sizes. We use the size 2^4 for our SR-LUT to reduce the LUT size, minimizing the loss of the original performance.

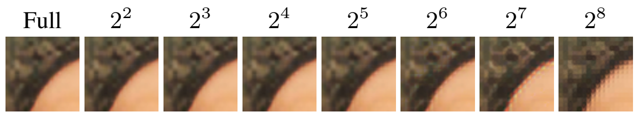

Figure 6. Visual comparison for different sampling interval sizes. Noticeable artifacts appear from the size 2^6 to 2^8.

5. Conclusion

We proposed a simple and practical single-image SR method by using LUT (SR-LUT). Our method is inherently faster as precomputed HR values are just retrieved from the SR-LUT and a few calculations is conducted for the final output. Compared to bicubic interpolation, our fast models (Ours-V and Ours-F) run faster while achieving better quantitative performance by a good margin, and our slow model (Ours-S) shows the better visual quality with a little more runtime. We believe our method is likely to be preferred in practical usages due to its speed and ease of implementation. In the future, using larger RF sizes to increase the quality and accelerating the interpolation step would make our approach more practical.

Acknowledgement This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2014-3-00123, Development of High Performance Visual BigData Discovery Platform for Large-Scale Realtime Data Analysis, and No. 2020-0-01361, Artificial Intelligence Graduate School Program (Yonsei University)).

References

[1] Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *CVPR Workshops*, July 2017. 6

[2] Namhyuk Ahn, Byungkon Kang, and Kyung-Ah Sohn. Fast, accurate, and lightweight super-resolution with cascading residual network. In *ECCV*, pages 252–268, 2018. 1, 3, 6, 7

[3] Simon Baker and Takeo Kanade. Hallucinating faces. In *Proceedings Fourth IEEE international conference on automatic face and gesture recognition (Cat. No. PR00580)*, pages 83–88. IEEE, 2000. 1

[4] Chenglong Bao, Jian-Feng Cai, and Hui Ji. Fast sparsity-based orthogonal dictionary learning for image restoration. In *ICCV*, pages 3384–3391, 2013. 2

[5] Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie line Alberi Morel. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. In *BMVC*, pages 135.1–135.10. BMVA Press, 2012. 1

[6] Hong Chang, Dit-Yan Yeung, and Yimin Xiong. Super-resolution through neighbor embedding. In *CVPR*, 2004. 6, 7

[7] Xiangxiang Chu, Bo Zhang, Hailong Ma, Ruijun Xu, and Qingyuan Li. Fast, accurate and lightweight super-resolution with neural architecture search. *arXiv preprint arXiv:1901.07261*, 2019. 3

[8] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Learning a deep convolutional network for image super-resolution. In *ECCV*, pages 184–199, 2014. 1

[9] Chao Dong, Chen Change Loy, and Xiaoou Tang. Accelerating the super-resolution convolutional neural network. In *ECCV*, pages 391–407. Springer, 2016. 1, 3, 4, 6, 7

[10] William T Freeman, Thouis R Jones, and Egon C Pasztor. Example-based super-resolution. *IEEE Computer graphics and Applications*, 22(2):56–65, 2002. 1

[11] William T Freeman, Egon C Pasztor, and Owen T Carmichael. Learning low-level vision. *IJCV*, 40(1):25–47, 2000. 1

[12] Daniel Glasner, Shai Bagon, and Michal Irani. Super-resolution from a single image. In *ICCV*, pages 349–356. IEEE, 2009. 1

[13] Muhammad Haris, Gregory Shakhnarovich, and Norimichi Ukita. Deep back-projection networks for super-resolution. In *CVPR*, pages 1664–1673, 2018. 1

[14] Jia-Bin Huang, Abhishek Singh, and Narendra Ahuja. Single image super-resolution from transformed self-exemplars. In *CVPR*, pages 5197–5206, 2015. 6

[15] Zheng Hui, Xiumei Wang, and Xinbo Gao. Fast and accurate single image super-resolution via information distillation network. In *CVPR*, pages 723–731, 2018. 3

[16] Andrey Ignatov, Radu Timofte, Andrei Kulik, Seungsoo Yang, Ke Wang, Felix Baum, Max Wu, Lirong Xu, and Luc Van Gool. Ai benchmark: All about deep learning on smartphones in 2019. In *ICCV Workshops*, pages 3617–3635. IEEE, 2019. 2

[17] Hakki Can Karaimer and Michael S Brown. A software platform for manipulating the camera imaging pipeline. In *ECCV*, pages 429–444. Springer, 2016. 2

[18] James M Kasson, Sigfredo I Nin, Wil Plouffe, and James Lee Hafner. Performing color space conversions with three-dimensional linear interpolation. *Journal of Electronic Imaging*, 4(3):226–251, 1995. 5

[19] Robert Keys. Cubic convolution interpolation for digital image processing. *IEEE transactions on acoustics, speech, and signal processing*, 29(6):1153–1160, 1981. 1

[20] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *CVPR*, pages 1646–1654, 2016. 1

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 6

[22] Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. Deep laplacian pyramid networks for fast and accurate super-resolution. In *CVPR*, pages 624–632, 2017. 1

[23] Royson Lee, Łukasz Dudziak, Mohamed Abdelfattah, Stylianos I Venieris, Hyeji Kim, Hongkai Wen, and Nicholas D Lane. Journey towards tiny perceptual super-resolution. In *ECCV*, 2020. 3

[24] Huixia Li, Chenqian Yan, Shaohui Lin, Xiawu Zheng, Baochang Zhang, Fan Yang, and Rongrong Ji. Pams: Quantized super-resolution via parameterized max scale. In *ECCV*, 2020. 3

[25] Zhen Li, Jinglei Yang, Zheng Liu, Xiaomin Yang, Gwanggil Jeon, and Wei Wu. Feedback network for image super-resolution. In *CVPR*, pages 3867–3876, 2019. 1

[26] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *CVPR Workshops*, pages 136–144, 2017. 1, 4

[27] Hai Ting Lin, Zheng Lu, Seon Joo Kim, and Michael S Brown. Nonuniform lattice regression for modeling the camera imaging pipeline. In *ECCV*, pages 556–568. Springer, 2012. 5

[28] Xiaotong Luo, Yuan Xie, Yulun Zhang, Yanyun Qu, Cuihua Li, and Yun Fu. Latticenet: Towards lightweight image super-resolution with lattice block. In *ECCV*, 2020. 3

[29] David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *ICCV*, volume 2, pages 416–423. IEEE, 2001. 6

[30] Yusuke Matsui, Kota Ito, Yuji Aramaki, Azuma Fujimoto, Toru Ogawa, Toshihiko Yamasaki, and Kiyoharu Aizawa. Sketch-based manga retrieval using manga109 dataset. *Multimedia Tools and Applications*, 76(20):21811–21838, 2017. 6

[31] Yiqun Mei, Yuchen Fan, Yuqian Zhou, Lichao Huang, Thomas S Huang, and Honghui Shi. Image super-resolution with cross-scale non-local attention and exhaustive self-exemplars mining. In *CVPR*, pages 5690–5699, 2020. 1

[32] Rang MH Nguyen and Michael S Brown. Raw image reconstruction using a self-contained srgb-jpeg image with only 64 kb overhead. In *CVPR*, pages 1655–1663, 2016. 5

[33] Ben Niu, Weilei Wen, Wenqi Ren, Xiangde Zhang, Lianping Yang, Shuzhen Wang, Kaihao Zhang, Xiaochun Cao, and Haifeng Shen. Single image super-resolution via a holistic attention network. In *CVPR*, 2020. 1

[34] Matt Pharr and Randima Fernando. *Gpu gems 2: programming techniques for high-performance graphics and general-purpose computation*. Addison-Wesley Professional, 2005. 2

[35] Guoping Qiu. Interresolution look-up table for improved spatial magnification of image. *Journal of Visual Communication and Image Representation*, 11(4):360–373, 2000. 2

[36] Yajun Qiu, Ruxin Wang, Dapeng Tao, and Jun Cheng. Embedded block residual network: A recursive restoration model for single-image super-resolution. In *ICCV*, pages 4180–4189, 2019. 1

[37] Yaniv Romano, John Isidoro, and Peyman Milanfar. Raisr: rapid and accurate image super resolution. *IEEE Transactions on Computational Imaging*, 3(1):110–125, 2016. 2

[38] Samuel Schulter, Christian Leistner, and Horst Bischof. Fast and accurate image upscaling with super-resolution forests. In *CVPR*, pages 3791–3799, 2015. 2

[39] Wenzhe Shi, Jose Caballero, Ferenc Huszar, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *CVPR*, pages 1874–1883, 2016. 3, 4

[40] Dehua Song, Chang Xu, Xu Jia, Yiyi Chen, Chunjing Xu, and Yunhe Wang. Efficient residual dense block search for image super-resolution. In *AAAI*, pages 12007–12014, 2020. 3

[41] Qiang Song, Ruiqin Xiong, Dong Liu, Zhiwei Xiong, Feng Wu, and Wen Gao. Fast image super-resolution via local adaptive gradient field sharpening transform. *IEEE TIP*, 27(4):1966–1980, 2018. 2

[42] Radu Timofte, Vincent De Smet, and Luc Van Gool. Anchored neighborhood regression for fast example-based super-resolution. In *ICCV*, pages 1920–1927, 2013. 1, 2, 6, 7

[43] Radu Timofte, Vincent De Smet, and Luc Van Gool. A+: Adjusted anchored neighborhood regression for fast super-resolution. In *ACCV*, pages 111–126. Springer, 2014. 1, 2, 6, 7

[44] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. Esrgan: Enhanced super-resolution generative adversarial networks. In *ECCV Workshops*, September 2018. 1, 6, 7

[45] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE TIP*, 13(4):600–612, 2004. 6

[46] Jingwei Xin, Nannan Wang, Xinrui Jiang, Jie Li, Heng Huang, and Xinbo Gao. Binarized neural network for single image super resolution. In *ECCV*, 2020. 3

[47] Jianchao Yang, Zhe Lin, and Scott Cohen. Fast image super-resolution based on in-place example regression. In *CVPR*, pages 1059–1066, 2013. 1

[48] Jianchao Yang, John Wright, Thomas S Huang, and Yi Ma. Image super-resolution via sparse representation. *IEEE TIP*, 19(11):2861–2873, 2010. 1, 2

[49] Hui Zeng, Jianrui Cai, Lida Li, Zisheng Cao, and Lei Zhang. Learning image-adaptive 3d lookup tables for high performance photo enhancement in real-time. *IEEE TPAMI*, 2020. 3

[50] Roman Zeyde, Michael Elad, and Matan Protter. On single image scale-up using sparse-representations. In *International conference on curves and surfaces*, pages 711–730. Springer, 2010. 1, 2, 6, 7

[51] Kai Zhang, Martin Danelljan, Yawei Li, Radu Timofte, Jie Liu, Jie Tang, Gangshan Wu, Yu Zhu, Xiangyu He, Wenjie Xu, et al. Aim 2020 challenge on efficient super-resolution: Methods and results. In *ECCV Workshops*, 2020. 3

[52] Yulun Zhang, Kunpeng Li, Kai Li, Lichen Wang, Bineng Zhong, and Yun Fu. Image super-resolution using very deep residual channel attention networks. In *ECCV*, pages 286–301, 2018. 1, 4

[53] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. Residual dense network for image super-resolution. In *CVPR*, pages 2472–2481, 2018. 1