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], 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 ma-
ior 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 computations. 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 \times 720 output image from 320 \times 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.
To obtain model CARN by replacing the conventional convolutions and multiplication-addition operations of their original big models while minimizing the performance degradation compared to state-of-the-art performance by using a small model capacity.

3.2. 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. 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 [25] 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 a desktop and 371ms on a Samsung Galaxy S7 smartphone. 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. 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 upsampling 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 (256 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...
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:

\[
\hat{y}_i = \frac{1}{4} \sum_{j=0}^{3} R_j^{-1}(f(R_j(x_i))),
\]

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:

\[
\sum_i \ell(\hat{y}_i, y_i),
\]

where \( y_i \) is GT target patch and \( \ell \) 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).

<table>
<thead>
<tr>
<th>RF</th>
<th>LUT</th>
<th>Full size</th>
<th>Sampled size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 pixel</td>
<td>1D</td>
<td>4KB</td>
<td>272B</td>
</tr>
<tr>
<td>2 pixels</td>
<td>2D</td>
<td>1MB</td>
<td>4.516KB</td>
</tr>
<tr>
<td>3 pixels</td>
<td>3D</td>
<td>256MB</td>
<td>76.766KB</td>
</tr>
<tr>
<td>4 pixels</td>
<td>4D</td>
<td>64GB</td>
<td>1.274MB</td>
</tr>
<tr>
<td>5 pixels</td>
<td>5D</td>
<td>16TB</td>
<td>21.665MB</td>
</tr>
<tr>
<td>n pixels</td>
<td>nD</td>
<td>((2^n)^n \times r^2B)</td>
<td>((2^4 + 1)^n \times r^2B)</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>LUT</th>
<th>Interpolation</th>
<th>Multiplications</th>
<th>Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D</td>
<td>Bilinear</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Triangular</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3D</td>
<td>Trilinear</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Tetrahedral</td>
<td>4</td>
<td>2.5</td>
</tr>
<tr>
<td>4D</td>
<td>Tetralinear</td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>4-simplex</td>
<td>5</td>
<td>4.5</td>
</tr>
</tbody>
</table>

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. 2e). 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 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, trilinear, 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 \( I_0 = 24 \) 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.

![Figure 3. An example of triangular interpolation for a sampled 2D LUT with the sampling interval of 2.](image)

<table>
<thead>
<tr>
<th>Condition</th>
<th>( w_0 )</th>
<th>( w_1 )</th>
<th>( w_2 )</th>
<th>( w_4 )</th>
<th>( O_1 )</th>
<th>( O_2 )</th>
<th>( O_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( L_x &gt; L_y &gt; L_z &gt; L_t )</td>
<td>( W ) &amp;</td>
<td>( W ) &amp;</td>
<td>( W ) &amp;</td>
<td>( W ) &amp;</td>
<td>( LUT[1][4] )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( L_x &gt; L_y &gt; L_z &gt; L_t )</td>
<td>( L_x ) &amp;</td>
<td>( L_y ) &amp;</td>
<td>( L_z ) &amp;</td>
<td>( L_t )</td>
<td>( LUT[1][4] )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( L_x &gt; L_y &gt; L_z &gt; L_t )</td>
<td>( L_x ) &amp;</td>
<td>( L_y ) &amp;</td>
<td>( L_z ) &amp;</td>
<td>( L_t )</td>
<td>( LUT[1][4] )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( L_x &gt; L_y &gt; L_z &gt; L_t )</td>
<td>( L_x ) &amp;</td>
<td>( L_y ) &amp;</td>
<td>( L_z ) &amp;</td>
<td>( L_t )</td>
<td>( LUT[1][4] )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.
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: $V = (w_0P_{00} + w_1P_{01} + w_2P_{02})/W$.

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 $I_0, I_1, I_2, I_3$ 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:

$$V = \frac{1}{W} \sum_{i=0}^{4} w_iO_i, \quad (3)$$

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 upsampling factor to 4 (i.e., $r = 4$). Our SR deep model is trained for 2 $\times$ $10^6$ 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 $\times$ 720 output RGB image from 320 $\times$ 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). Similarly, 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 ($1 \times 2$ and $1 \times 3$ respectively) can consider more pixels horizontally and vertically, whereas the kernel of Ours-S ($2 \times 2$) 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.
Table 4. Quantitative comparisons on 5 common single-image SR test sets 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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Runtime</th>
<th>Size</th>
<th>Set5 PSNR</th>
<th>SSIM</th>
<th>Set14 PSNR</th>
<th>SSIM</th>
<th>BSDS100 PSNR</th>
<th>SSIM</th>
<th>Urban100 PSNR</th>
<th>SSIM</th>
<th>Manga109 PSNR</th>
<th>SSIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nearest</td>
<td>4ms</td>
<td>-</td>
<td>26.25</td>
<td>0.7372</td>
<td>24.65</td>
<td>0.6529</td>
<td>25.03</td>
<td>0.6293</td>
<td>22.17</td>
<td>0.6154</td>
<td>23.45</td>
<td>0.7414</td>
</tr>
<tr>
<td>Bilinear</td>
<td>16ms</td>
<td>-</td>
<td>27.55</td>
<td>0.7884</td>
<td>25.42</td>
<td>0.6792</td>
<td>25.54</td>
<td>0.6460</td>
<td>22.69</td>
<td>0.6346</td>
<td>24.21</td>
<td>0.7666</td>
</tr>
<tr>
<td>Bicubic</td>
<td>60ms</td>
<td>-</td>
<td>28.42</td>
<td>0.8101</td>
<td>26.00</td>
<td>0.7023</td>
<td>25.96</td>
<td>0.6672</td>
<td>23.14</td>
<td>0.6574</td>
<td>24.91</td>
<td>0.7871</td>
</tr>
<tr>
<td>Ours-V</td>
<td>15ms</td>
<td>1MB</td>
<td>29.22</td>
<td>0.8304</td>
<td>26.65</td>
<td>0.7258</td>
<td>26.33</td>
<td>0.6880</td>
<td>23.68</td>
<td>0.6852</td>
<td>26.30</td>
<td>0.8246</td>
</tr>
<tr>
<td>Ours-F</td>
<td>34ms</td>
<td>77KB</td>
<td>29.77</td>
<td>0.8429</td>
<td>26.99</td>
<td>0.7372</td>
<td>26.57</td>
<td>0.6990</td>
<td>23.94</td>
<td>0.7666</td>
<td>26.87</td>
<td>0.8367</td>
</tr>
<tr>
<td>Ours-S</td>
<td>91ms</td>
<td>1.274MB</td>
<td>29.82</td>
<td>0.8478</td>
<td>27.01</td>
<td>0.7355</td>
<td>26.53</td>
<td>0.6953</td>
<td>24.02</td>
<td>0.6990</td>
<td>26.80</td>
<td>0.8380</td>
</tr>
<tr>
<td>NE + LLE [6]</td>
<td>7016ms</td>
<td>1.43MB</td>
<td>29.62</td>
<td>0.8404</td>
<td>26.82</td>
<td>0.7346</td>
<td>26.49</td>
<td>0.6970</td>
<td>23.84</td>
<td>0.6942</td>
<td>26.10</td>
<td>0.8195</td>
</tr>
<tr>
<td>Zeyde et al. [50]</td>
<td>8797ms</td>
<td>1.43MB</td>
<td>26.69</td>
<td>0.8429</td>
<td>26.90</td>
<td>0.7354</td>
<td>26.53</td>
<td>0.6968</td>
<td>23.90</td>
<td>0.6962</td>
<td>26.24</td>
<td>0.8241</td>
</tr>
<tr>
<td>ANR [42]</td>
<td>1715ms</td>
<td>1.43MB</td>
<td>29.70</td>
<td>0.8422</td>
<td>26.86</td>
<td>0.7368</td>
<td>26.52</td>
<td>0.6992</td>
<td>23.89</td>
<td>0.6964</td>
<td>26.18</td>
<td>0.8214</td>
</tr>
<tr>
<td>A+ [43]</td>
<td>1748ms</td>
<td>15.171MB</td>
<td>30.27</td>
<td>0.8602</td>
<td>27.30</td>
<td>0.7498</td>
<td>26.73</td>
<td>0.7088</td>
<td>24.33</td>
<td>0.7189</td>
<td>26.91</td>
<td>0.8480</td>
</tr>
<tr>
<td>FSRCNN [9]</td>
<td>371ms</td>
<td>12K†</td>
<td>30.71</td>
<td>0.8656</td>
<td>27.60</td>
<td>0.7543</td>
<td>26.96</td>
<td>0.7129</td>
<td>24.61</td>
<td>0.7263</td>
<td>27.91</td>
<td>0.8587</td>
</tr>
<tr>
<td>CARN-M [2]</td>
<td>4955ms</td>
<td>412K†</td>
<td>31.82</td>
<td>0.8898</td>
<td>28.29</td>
<td>0.7747</td>
<td>27.42</td>
<td>0.7305</td>
<td>25.62</td>
<td>0.7694</td>
<td>29.85</td>
<td>0.8993</td>
</tr>
<tr>
<td>RRDB [44]</td>
<td>31717ms</td>
<td>16698K†</td>
<td>32.68</td>
<td>0.8999</td>
<td>28.88</td>
<td>0.7891</td>
<td>27.82</td>
<td>0.7891</td>
<td>27.02</td>
<td>0.8146</td>
<td>31.57</td>
<td>0.9185</td>
</tr>
</tbody>
</table>

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.

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 $1 \times 2$ kernel (Ours-V) exploits total 5 input pixels by using the rotational ensemble. Similarly, the RF size of 3 with $1 \times 3$ kernel (Ours-F) covers 9, and the RF size of 4 with $1 \times 4$ 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 $1 \times 3$ kernel (Ours-F) covers 9, and the RF size of 4 with $1 \times 4$ 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.

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.
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 $1 \times 4$ and $2 \times 2$ respectively. They show almost equivalent performance in quantitative comparison. However, Ours-S shows more visually pleasing results as shown in Fig. 5 as $2 \times 2$ 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 original 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 $2^4$ 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.

**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 performance and the speed of the methods 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


[28] Xiaotong Luo, Yuan Xie, Yulun Zhang, Yanyun Qu, Cuixia Li, and Yun Fu. Latticenet: Towards lightweight image super-resolution. In CVPR Workshops, pages 1655–1663, 2016. 5


