# Learning Steerable Function for Efficient Image Resampling Supplementary Document

Jiacheng Li[1*] Chang Chen[2*] Wei Huang[1]
Zhiqiang Lang[2] Fenglong Song[2] Youliang Yan[2] Zhiwei Xiong[1†]
[1]University of Science and Technology of China [2]Huawei Noah's Ark Lab
{jclee,weih527}@mail.ustc.edu.cn zwxiong@ustc.edu.cn
{chenchang25,langzhiqiang,songfenglong,yanyouliang}@huawei.com


Figure 1. Continuous upsampling results of LeRF. More results are available on our project page: https://lerf.pages.dev.

In this supplementary document and the accompanying supplementary videos, we include the following contents:

- continuous resampling results (Sec. 1),
- more comparisons (Sec. 2),
- the details on LUT acceleration, evaluation, and implementation (Sec. 3),
- more quantitative and qualitative results (Sec. 4).

## 1. Continuous Resampling Results

In Fig. 1, we provide an example of continuous upsampling results of LeRF. Besides, in our project page (https://lerf.pages.dev), we present two comparison videos of continuous resampling results. In the first one, we compare the visual results of LeRF and bicubic side-by-side for a continuous upsampling range, *i.e.*, from $\times 1$ to $\times 8$. In the second one, we compare the continuous resampling results of LeRF and bicubic for general homographic transformations, including asymmetric upsampling, downsampling, sheering, and rotation. As can be seen, LeRF produces more visually pleasing results than the widely-used bicubic interpolation.

| | RunTime | MACs | $\times 2$ | $\times 3$ | $\times 4$ |
|---|---|---|---|---|---|
| LeRF (Ours) | 110 | 57.94M | 35.71 | 32.02 | 30.15 |
| SR-LUT ($\times 2$ oversample) | 149 | 57.66M | 35.53 | 31.91 | 29.75 |
| SR-LUT ($\times 4$ oversample) | 207 | 74.94M | 35.60 | 31.94 | 29.80 |

Table 1. Performance and efficiency comparison with a cascaded oversampling solution. RunTime in ms and evaluated on a mobile phone. Performance reported in PSNR for $\times 2$, $\times 3$, and $\times 4$ upsampling on the Set5 dataset.

## 2. More Comparisons

**Comparison with cascaded oversampling solutions.** We compare LeRF with alternative solutions to achieve arbitrary-scale upsampling, where an input image is firstly over-upsampled to $\times 2$ or $\times 4$ scale by a trained SR-LUT, and then downsampled to the actual target resolution with bicubic interpolation. As listed in Table 1, these cascaded oversampling solutions yield comparable performance, but at a cost of waste of computational resources due to oversampling.

*Equal contribution. †Corresponding author. This work was done when Jiacheng Li was a research intern at Huawei Noah's Ark Lab.

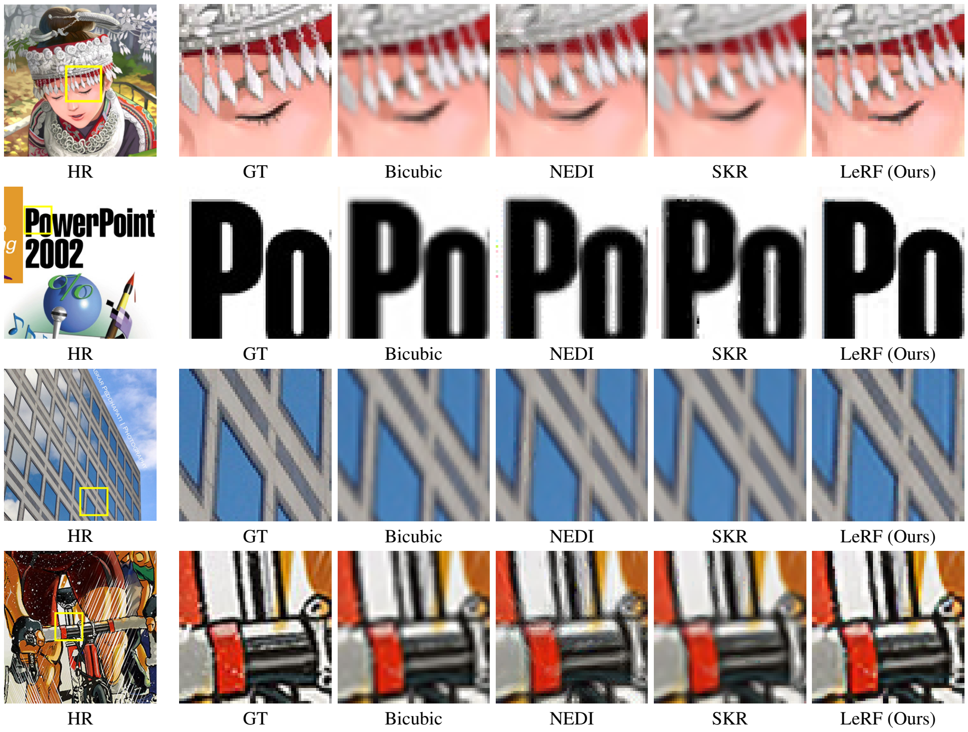


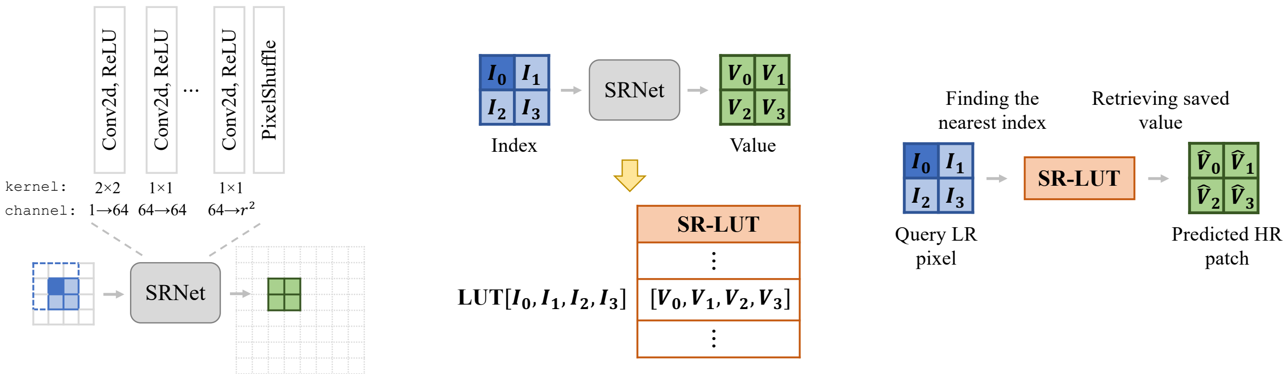

Figure 2. Qualitative comparison with rule-based adaptive resampling methods for $\times 2$ upsampling. We also include bicubic as a companion. From top to bottom, the example images are: *comic* (Set14), *ppt3* (Set14), *img035* (Urban100), and *KyokugenCyclone* (Manga109). Best view on screen and in color.

**Comparison with rule-based adpative resampling methods.** In Fig. 2, we compare LeRF with two rule-based adaptive resampling methods, *i.e.*, NEDI [6] and SKR [9]. They estimate the resampling weights according to hand-designed rules based on local gradients. As can be seen, LeRF obtains better visual quality than these rule-based adaptive resampling methods, showing the advantage of extracting structural priors in a learning-based way.

## 3. More Implementation Details

**LUT acceleration.** Here, we review the detailed process of LUT acceleration in SR-LUT [5]. As illustrated in Fig.3, in SR-LUT, the authors first train a deep super-resolution network (SRNet). Then, they traverse all possible input LR patches, *i.e.*, $[I_0, I_1, I_2, I_3]$, and pre-compute the corresponding HR patches, *i.e.*, $[V_0, V_1, V_2, V_3]$. The anchor pixel $I_0$ and its surrounding pixels ($I_2$, $I_3$, and $I_4$), are saved as the index, while the corresponding HR patch is saved as the value. Finally, at inference time, the HR predictions are calculated by finding the nearest index to the query LR pixels in indices and retrieving corresponding saved HR values. The four HR pixels replace the original anchor pixel $I_0$, thus resulting in $\times 2$ upsampling of an input image. In SR-LUT, the LUT is uniformly sampled with an interval of $2^4$ to reduce storage. In our work, we follow the above process to accelerate the DNN with LUT. Differently, instead of saving HR patches, our LUTs save the hyper-parameters that determine the orientations of resampling functions. For our DNN, each branch can be treated as an SRNet and thus accelerated by a LUT with the same indexing pattern. In total, we use three LUTs for the pre-filter stage ($S$, $C$, and $X$ patterns) and six LUTs for hyper-parameter prediction ($S$, $S'$, $C$, $C'$, $X$, and $X'$ patterns). We adopt the rotation ensemble strategy in the pre-filter stage, and the proposed directional ensemble strategy for hyper-parameter prediction.

**Efficiency evaluation.** For interpolation methods, SR-LUT* and LeRF, the running time is evaluated on a OnePlus 7 Pro mobile phone with a Qualcomm Snapdragon 855 CPU. For RAISR* [7], Meta-SR [4], and LIIF [2], that is evaluated on a desktop computer with an Intel Xeon Gold 6278C 2.60GHz CPU. We implement LeRF and interpolation methods under the Java `IntStream.parallel()` API, and the number of parallel threads is set to the total pixels (*i.e.*, $h \times w$) of an input image. We estimate the MACs of LeRF and interpolation methods based on the size of the target image and the operations needed per target pixel. Specifically, for the `exp()` operation in LeRF and `sinc()` operation in Lanczos interpolation, we count their MAC as 4, according to the theoretical estimation in [1] and the decimal precision of `float` type number. For LeRF, the LUT




(a) Training SRNet (b) Traversing and pre-computing index-value pairs (c) LUT retrieval instead of network computation

Figure 3. The detailed process of LUT acceleration in SR-LUT [5]. (a) In SR-LUT, the authors first train an SRNet. (b) Then, they traverse all possible input LR patches, pre-compute corresponding HR patches, and save them as index-value pairs in a LUT. (c) At inference time, HR patches are predicted by finding the nearest indices to the query LR pixels and retrieving the saved HR values. $r$ denotes the upsampling scale ($r = 2$ here). This figure is reproduced according to Fig. 2 in the SR-LUT paper [5].

execution and interpolation with the predicted resampling functions account for 17.39M and 40.55M MACs, respectively. The LUT execution mainly involves `int` type operations, which contributes to efficiency advantage. For DNN-based methods, we calculate their MACs with `torchinfo`[1]. We estimate the LUT size according to Table 1 in the SR-LUT paper [5].

**Implementation.** For interpolation methods, we adopt the implementation[2] presented in [8]. We adopt the official implementations of SR-LUT[3], Meta-SR[4], and LIIF[5]. We adopt the third-party RAISR implementation[6] presented in [3].

## 4. More Results

To further quantitatively evaluate the perceptual quality of LeRF, we adopt the LPIPS [10] metric and list the results for upsampling in Table 2. As can be seen, LeRF outperforms interpolation methods by a large margin and achieves comparable (sometimes even better) performance with DNN-based methods, showing its clear advantage in perceptual quality.

Besides, in Table 3, we provide the SSIM results for LeRF and comparison methods for arbitrary-scale upsampling. Although RAISR* performs slightly better than our method occasionally, the fused results from multiple models are hard to achieve in practice. In Fig. 4 and Fig. 5, we provide additional examples of qualitative comparison.

[1] https://github.com/TylerYep/torchinfo
[2] https://github.com/assafshocher/ResizeRight
[3] https://github.com/yhjo09/SR-LUT
[4] https://github.com/XuecaiHu/Meta-SR-Pytorch
[5] https://github.com/yinboc/liif
[6] https://github.com/JalaliLabUCLA/Jalali-Lab-Implementation-of-RAISR

## References

[1] J. M. Borwein and P. B. Borwein. On the complexity of familiar functions and numbers. *SIAM Review*, 30(4):589–601, 1988. 2

[2] Yinbo Chen, Sifei Liu, and Xiaolong Wang. Learning continuous image representation with local implicit image function. In *CVPR*, 2021. 2, 4

[3] Sifeng He and Bahram Jalali. Brain MRI image super resolution using phase stretch transform and transfer learning. *arxiv*, 1807.11643, 2018. 3

[4] Xuecai Hu, Haoyuan Mu, Xiangyu Zhang, Zilei Wang, Tieniu Tan, and Jian Sun. Meta-sr: A magnification-arbitrary network for super-resolution. In *CVPR*, 2019. 2, 4

[5] Younghyun Jo and Seon Joo Kim. Practical single-image super-resolution using look-up table. In *CVPR*, 2021. 2, 3, 4

[6] Xin Li and Michael T. Orchard. New edge-directed interpolation. *IEEE Trans. Image Process.*, 10(10):1521–1527, 2001. 2

[7] Yaniv Romano, John Isidoro, and Peyman Milanfar. RAISR: rapid and accurate image super resolution. *IEEE Trans. Computational Imaging*, 3(1):110–125, 2017. 2, 4

[8] Assaf Shocher, Ben Feinstein, Niv Haim, and Michal Irani. From discrete to continuous convolution layers. *arxiv*, 2006.11120, 2020. 3

[9] Hiroyuki Takeda, Sina Farsiu, and Peyman Milanfar. Kernel regression for image processing and reconstruction. *IEEE Trans. Image Process.*, 16(2):349–366, 2007. 2

[10] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 3

| Method | Set5 | | | Set14 | | | BSDS100 | | | Urban100 | | | Manga109 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\times 2$ | $\times 3$ | $\times 4$ | $\times 2$ | $\times 3$ | $\times 4$ | $\times 2$ | $\times 3$ | $\times 4$ | $\times 2$ | $\times 3$ | $\times 4$ | $\times 2$ | $\times 3$ | $\times 4$ |
| Bilinear | 0.1674 | 0.2421 | 0.3572 | 0.2444 | 0.3433 | 0.4645 | 0.3181 | 0.4228 | 0.5503 | 0.2640 | 0.3764 | 0.4965 | 0.1502 | 0.2403 | 0.3481 |
| Bicubic | 0.1261 | 0.2508 | 0.3461 | 0.1958 | 0.3603 | 0.4573 | 0.2643 | 0.4451 | 0.5432 | 0.2203 | 0.3947 | 0.4932 | 0.1107 | 0.2416 | 0.3340 |
| Lanczos3 | 0.1349 | 0.2527 | 0.3501 | 0.2177 | 0.3649 | 0.4628 | 0.2968 | 0.4538 | 0.5530 | 0.2476 | 0.3953 | 0.4940 | 0.1209 | 0.2389 | 0.3320 |
| RAISR* [7] | 0.0784 | 0.1669 | 0.2421 | 0.1418 | 0.2784 | 0.3588 | 0.1984 | 0.3651 | 0.4397 | 0.1567 | 0.3062 | 0.3915 | 0.0582 | 0.1591 | 0.2461 |
| SR-LUT* [5] | 0.0882 | 0.2206 | 0.3186 | 0.1499 | 0.3364 | 0.4359 | 0.2138 | 0.4197 | 0.5177 | 0.1857 | 0.3953 | 0.4728 | 0.0671 | 0.1877 | 0.2675 |
| LeRF (Ours) | **0.0504** | **0.1062** | **0.1830** | **0.0879** | **0.2022** | **0.3023** | **0.1472** | **0.2906** | **0.3995** | **0.1096** | **0.2377** | **0.3374** | **0.0325** | **0.0992** | **0.1699** |
| MetaSR [4] | 0.0576 | 0.1213 | 0.1706 | 0.0952 | 0.2083 | 0.2886 | 0.1490 | 0.2857 | 0.3766 | 0.0599 | 0.1452 | 0.2179 | 0.0236 | 0.0643 | 0.1052 |
| LIIF [2] | 0.0583 | 0.1223 | 0.1714 | 0.0953 | 0.2090 | 0.2918 | 0.1557 | 0.2907 | 0.3793 | 0.0622 | 0.1470 | 0.2216 | 0.0246 | 0.0656 | 0.1083 |

Table 2. Quantitative comparison in LPIPS for arbitrary-scale upsampling. * denotes that we combine fixed-scale super-resolution methods with bicubic interpolation to achieve arbitrary-scale upsampling. Lower is better. The best results are **highlighted**.

| Method | Set5 | | | | Set14 | | | | BSDS100 | | | | Urban100 | | | | Manga109 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\frac{\times 1.5}{\times 1.5}$ | $\frac{\times 1.5}{\times 2.0}$ | $\frac{\times 2.0}{\times 2.0}$ | $\frac{\times 2.0}{\times 2.4}$ | $\frac{\times 1.5}{\times 1.5}$ | $\frac{\times 1.5}{\times 2.0}$ | $\frac{\times 2.0}{\times 2.0}$ | $\frac{\times 2.0}{\times 2.4}$ | $\frac{\times 1.5}{\times 1.5}$ | $\frac{\times 1.5}{\times 2.0}$ | $\frac{\times 2.0}{\times 2.0}$ | $\frac{\times 2.0}{\times 2.4}$ | $\frac{\times 1.5}{\times 1.5}$ | $\frac{\times 1.5}{\times 2.0}$ | $\frac{\times 2.0}{\times 2.0}$ | $\frac{\times 2.0}{\times 2.4}$ | $\frac{\times 1.5}{\times 1.5}$ | $\frac{\times 1.5}{\times 2.0}$ | $\frac{\times 2.0}{\times 2.0}$ | $\frac{\times 2.0}{\times 2.4}$ |
| Nearest | 0.9188 | 0.9086 | 0.9001 | 0.8747 | 0.8774 | 0.8631 | 0.8466 | 0.8168 | 0.8626 | 0.8456 | 0.8255 | 0.7944 | 0.8595 | 0.8405 | 0.8213 | 0.7871 | 0.9281 | 0.9187 | 0.9103 | 0.8829 |
| Bilinear | 0.9491 | 0.9287 | 0.9120 | 0.8991 | 0.9052 | 0.8745 | 0.8411 | 0.8240 | 0.8868 | 0.8497 | 0.8110 | 0.7919 | 0.8844 | 0.8451 | 0.8094 | 0.7886 | 0.9573 | 0.9327 | 0.9132 | 0.8963 |
| Bicubic | 0.9621 | 0.9446 | 0.9306 | 0.9174 | 0.9277 | 0.9004 | 0.8703 | 0.8515 | 0.9142 | 0.8802 | 0.8440 | 0.8218 | 0.9102 | 0.8741 | 0.8410 | 0.8173 | 0.9718 | 0.9514 | 0.9353 | 0.9175 |
| Lanczos2 | 0.9624 | 0.9450 | 0.9309 | 0.9176 | 0.9283 | 0.9012 | 0.8712 | 0.8522 | 0.9150 | 0.8811 | 0.8451 | 0.8227 | 0.9110 | 0.8750 | 0.8419 | 0.8180 | 0.9722 | 0.9519 | 0.9359 | 0.9179 |
| Lanczos3 | 0.9666 | 0.9499 | 0.9366 | 0.9234 | 0.9359 | 0.9096 | 0.8804 | 0.8614 | 0.9245 | 0.8911 | 0.8554 | 0.8327 | 0.9194 | 0.8836 | 0.8509 | 0.8268 | 0.9772 | 0.9578 | 0.9428 | 0.9246 |
| RAISR* [7] | 0.9516 | 0.9496 | **0.9481** | 0.9266 | 0.9121 | 0.9051 | 0.8990 | 0.8701 | 0.8987 | 0.8875 | 0.8787 | 0.8448 | 0.8956 | 0.8872 | 0.8831 | 0.8423 | 0.9621 | 0.9599 | **0.9593** | 0.9299 |
| SR-LUT* [5] | 0.9686 | 0.9528 | 0.9404 | 0.9278 | 0.9416 | 0.9164 | 0.8890 | 0.8707 | 0.9349 | 0.9043 | 0.8707 | 0.8490 | 0.9280 | 0.8945 | 0.8633 | 0.8399 | 0.9765 | 0.9581 | 0.9441 | 0.9263 |
| LeRF (Ours) | **0.9702** | **0.9574** | 0.9474 | **0.9376** | **0.9467** | **0.9243** | **0.8999** | **0.8830** | **0.9391** | **0.9104** | **0.8789** | **0.8580** | **0.9408** | **0.9117** | **0.8846** | **0.8626** | **0.9800** | **0.9676** | 0.9580 | **0.9462** |
| MetaSR [4] | 0.9785 | - | 0.9610 | - | 0.9602 | - | 0.9204 | - | 0.9544 | - | 0.9009 | - | 0.9653 | - | 0.9360 | - | 0.9904 | - | 0.9872 | - |
| LIIF [2] | 0.9784 | 0.9685 | 0.9611 | 0.9538 | 0.9600 | 0.9416 | 0.9210 | 0.9074 | 0.9545 | 0.9290 | 0.9011 | 0.8823 | 0.9692 | 0.9505 | 0.9352 | 0.9206 | 0.9903 | 0.9835 | 0.9781 | 0.9718 |

| Method | Set5 | | | | Set14 | | | | BSDS100 | | | | Urban100 | | | | Manga109 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\frac{\times 2.0}{\times 3.0}$ | $\frac{\times 3.0}{\times 3.0}$ | $\frac{\times 3.0}{\times 4.0}$ | $\frac{\times 4.0}{\times 4.0}$ | $\frac{\times 2.0}{\times 3.0}$ | $\frac{\times 3.0}{\times 3.0}$ | $\frac{\times 3.0}{\times 4.0}$ | $\frac{\times 4.0}{\times 4.0}$ | $\frac{\times 2.0}{\times 3.0}$ | $\frac{\times 3.0}{\times 3.0}$ | $\frac{\times 3.0}{\times 4.0}$ | $\frac{\times 4.0}{\times 4.0}$ | $\frac{\times 2.0}{\times 3.0}$ | $\frac{\times 3.0}{\times 3.0}$ | $\frac{\times 3.0}{\times 4.0}$ | $\frac{\times 4.0}{\times 4.0}$ | $\frac{\times 2.0}{\times 3.0}$ | $\frac{\times 3.0}{\times 3.0}$ | $\frac{\times 3.0}{\times 4.0}$ | $\frac{\times 4.0}{\times 4.0}$ |
| Nearest | 0.8507 | 0.8128 | 0.7701 | 0.7372 | 0.7904 | 0.7355 | 0.6952 | 0.6553 | 0.7665 | 0.7082 | 0.6701 | 0.6307 | 0.7564 | 0.7006 | 0.6562 | 0.6166 | 0.8572 | 0.8172 | 0.7748 | 0.7420 |
| Bilinear | 0.8774 | 0.8512 | 0.8155 | 0.7885 | 0.7987 | 0.7556 | 0.7185 | 0.6824 | 0.7652 | 0.7197 | 0.6837 | 0.6479 | 0.7591 | 0.7152 | 0.6730 | 0.6363 | 0.8706 | 0.8390 | 0.7984 | 0.7677 |
| Bicubic | 0.8951 | 0.8686 | 0.8355 | 0.8106 | 0.8241 | 0.7765 | 0.7410 | 0.7056 | 0.7916 | 0.7399 | 0.7049 | 0.6694 | 0.7848 | 0.7359 | 0.6954 | 0.6592 | 0.8902 | 0.8572 | 0.8183 | 0.7888 |
| Lanczos2 | 0.8952 | 0.8687 | 0.8355 | 0.8107 | 0.8247 | 0.7768 | 0.7412 | 0.7058 | 0.7923 | 0.7402 | 0.7053 | 0.6698 | 0.7854 | 0.7362 | 0.6956 | 0.6594 | 0.8905 | 0.8574 | 0.8184 | 0.7889 |
| Lanczos3 | 0.9010 | 0.8750 | 0.8413 | 0.8168 | 0.8338 | 0.7855 | 0.7493 | 0.7130 | 0.8018 | 0.7488 | 0.7130 | 0.6763 | 0.7940 | 0.7443 | 0.7030 | 0.6659 | 0.8967 | 0.8637 | 0.8238 | 0.7939 |
| RAISR* [7] | 0.9074 | 0.8968 | 0.8536 | 0.8431 | 0.8311 | 0.8087 | 0.7584 | 0.7357 | 0.8000 | 0.7729 | 0.7212 | 0.6963 | 0.8013 | 0.7796 | 0.7179 | 0.6983 | 0.9039 | 0.8918 | 0.8370 | 0.8240 |
| SR-LUT* [5] | 0.9061 | 0.8812 | 0.8498 | 0.8251 | 0.8442 | 0.7976 | 0.7620 | 0.7256 | 0.8188 | 0.7663 | 0.7298 | 0.6920 | 0.8076 | 0.7582 | 0.7168 | 0.6791 | 0.8985 | 0.8660 | 0.8261 | 0.7971 |
| LeRF (Ours) | **0.9189** | **0.8980** | **0.8732** | **0.8548** | **0.8573** | **0.8126** | **0.7816** | **0.7475** | **0.8278** | **0.7763** | **0.7412** | **0.7047** | **0.8309** | **0.7844** | **0.7462** | **0.7114** | **0.9246** | **0.9008** | **0.8708** | **0.8482** |
| MetaSR [4] | - | 0.9295 | - | 0.8985 | - | 0.8466 | - | 0.7876 | - | 0.8089 | - | 0.7416 | - | 0.8672 | - | 0.8049 | - | 0.9491 | - | 0.9175 |
| LIIF [2] | 0.9422 | 0.9291 | 0.9114 | 0.8981 | 0.8859 | 0.8472 | 0.8189 | 0.7878 | 0.8558 | 0.8101 | 0.7769 | 0.7425 | 0.8979 | 0.8664 | 0.8335 | 0.8043 | 0.9612 | 0.9487 | 0.9310 | 0.9169 |

Table 3. Quantitative comparison in SSIM for arbitrary-scale upsampling. $\frac{r_h}{r_w}$ denotes upsampling $r_h$ times along the short side and $r_w$ times along the long side. * denotes that we combine fixed-scale super-resolution methods with bicubic interpolation to achieve arbitrary-scale upsampling. The best and second best results are **highlighted** and underlined.

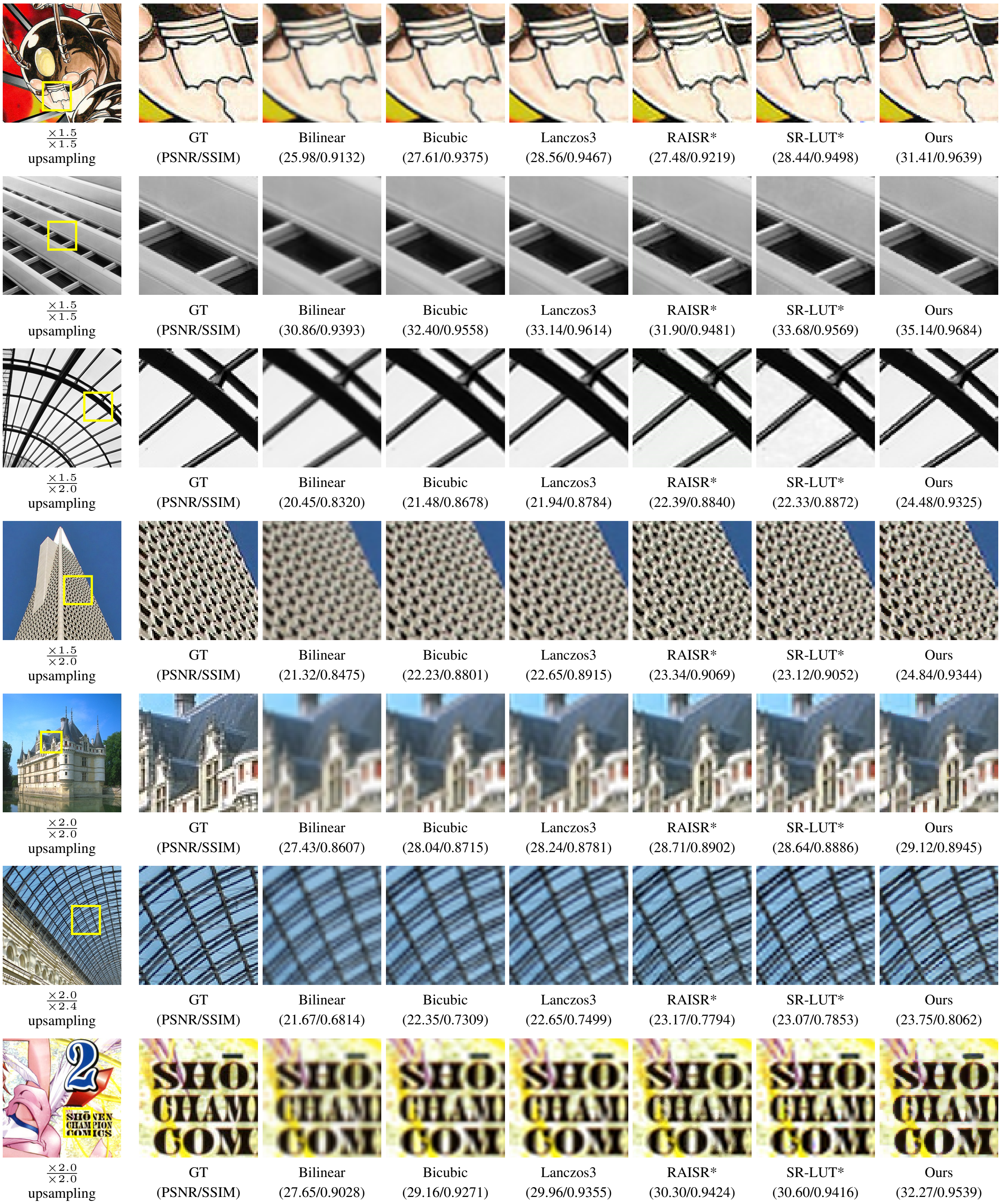



Figure 4. Additional qualitative comparison for arbitrary-scale upsampling. From top to bottom, the example images are: *TennenSenshiG* (Manga109), *img042* (Urabn100), *img072* (Urban100), *img048* (Urabn100), *102061* (BSDS100), *img008* (Urban100), and *DollGuan* (Manga109). Best view on screen and in color.

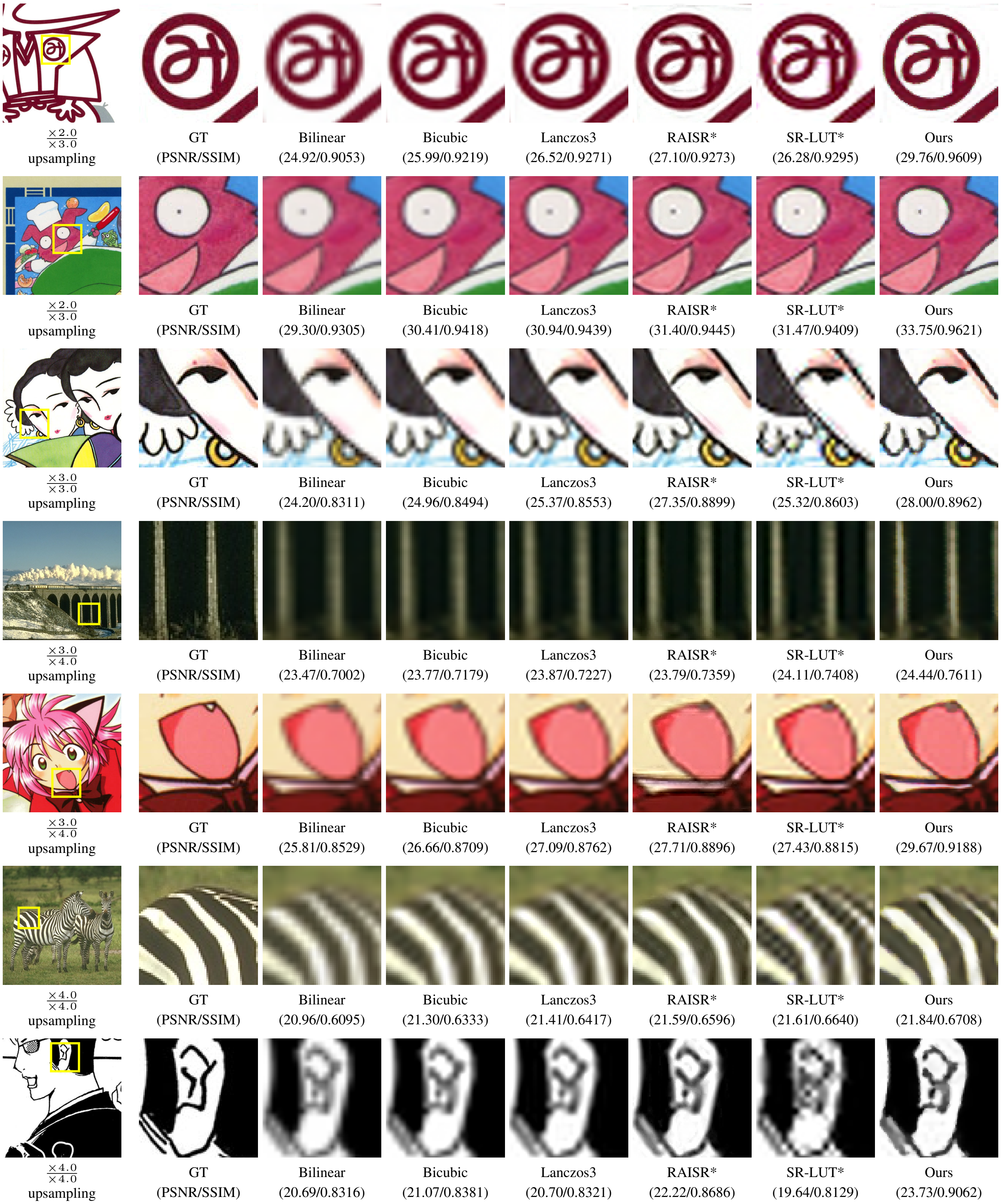



Figure 5. Additional qualitative comparison for arbitrary-scale upsampling. From top to bottom, the example images are: *Hamlet* (Manga109), *TapkunNoTanteisitsu* (Manga109), *TouyouKidan* (Manga109), *182053* (BSDS100), *Arisa* (Mang109), *253027* (BSDS100), and *TetsuSan* (Manga109). Best view on screen and in color.