1. Overview

In this work, we proposed a novel Efficient Super-Resolution Transformer (ESRT). In this supplementary material, we provide more experiments to further illustrate the effectiveness and advancement of ESRT. All codes available at https://github.com/luissen/ESRT.

1.1. Comparisons with Advanced SISR Models

1.1.1 Objective Evaluation

In TABLE 1, we compare our ESRT with more than 15 advanced SISR models. Most of them achieve the best results at the time with a well-designed lightweight network. Obviously, our ESRT achieves competitive results with a small amount of parameters. It can be seen that our ESRT performs much better than other models on Urban100 and Manga109 datasets. This is because there are many similar patches in each image of these datasets. Therefore, the introduced LTB in our ESRT can used to capture the long-term dependencies among these similar image patches and learn their relevance, thus future improve the performance of the model.

1.1.2 Subjective Evaluation

In Figure 1, we also provide more visual comparison between ESRT and other advanced SISR models. Obviously, SR images reconstructed by our ESRT contains more accurate texture details, especially in the edges and lines. It is worth noting that in the ×4 scale, the gap between ESRT and other SR models is more apparent. This benefits from the effectiveness of the proposed Efficient Transformer, which can learn more information from other clear areas. However, we also notice that there are still contains some error lines in our reconstructed images. This is because the LR image is a down-sampled image, some regional features in the image are severely damaged. For these areas, it is difficult for the model to match the available reference patches for its learning. Therefore, the reconstructed lines of these areas are not straight still.

1.2. Network Investigations

1.2.1 Study of Adaptive Residual Feature Block

In this work, we proposed a powerful Adaptive Residual Feature Block (ARFB) for feature extraction. In this part, we provide a detailed ablation study to validate the effectiveness of ARFB. Specifically, HPB is composed of an upper branch and a lower branch. Among them, the upper branch is used to extract high-frequency information and the lower branch is used to mine potential features. It is worth noting that all ARFBs share weights in the lower branch to reduce parameters. This means that this is a recursive component, which helps maximize the use of model parameters. In Table 2, we provide the impact of different ARFBs in the lower branch on model performance. According to the table, we can find: a) The introduced weight sharing strategy can further improve model performance; b) When the number of ARFB is increased, the model performance can be further improved; c) When the number of ARFBs increased to 6, the model performance no longer increased, and even a slight decrease. Therefore, we use 5 ARFBs in the lower branch on the final HPB to achieve the best results.

In the upper branch, we only use one ARFB to extract high-frequency information. In Table 3, we provide the impact of different ARFBs in the upper branch on model performance. Obviously, adding more ARFB will further improve model performance. However, it cannot be ignored that as the number of ARFB increases, the number of parameter of the model will also increase, which is not conducive to the construction of lightweight models. Meanwhile, the growth rate of model performance will also slow down. Therefore, we only use one ARFB in HPB to achieve a good balance between model size and performance.
<table>
<thead>
<tr>
<th>Method</th>
<th>Scale</th>
<th>Params</th>
<th>Set5 PSNR/SSIM</th>
<th>Set14 PSNR/SSIM</th>
<th>BSD100 PSNR/SSIM</th>
<th>Urban100 PSNR/SSIM</th>
<th>Manga109 PSNR/SSIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bicubic</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SRCNN [6]</td>
<td>8K</td>
<td>30.89 / 0.8682</td>
<td>27.55 / 0.7742</td>
<td>27.21 / 0.7358</td>
<td>24.46 / 0.7349</td>
<td>26.95 / 0.8556</td>
<td></td>
</tr>
<tr>
<td>FSRCCN [7]</td>
<td>13K</td>
<td>32.75 / 0.9090</td>
<td>29.30 / 0.8215</td>
<td>28.41 / 0.7863</td>
<td>26.24 / 0.7989</td>
<td>30.48 / 0.9117</td>
<td></td>
</tr>
<tr>
<td>VDSR [11]</td>
<td>666K</td>
<td>33.66 / 0.9213</td>
<td>29.77 / 0.8314</td>
<td>28.82 / 0.7976</td>
<td>27.14 / 0.8279</td>
<td>32.01 / 0.9340</td>
<td></td>
</tr>
<tr>
<td>DRCN [12]</td>
<td>1,774K</td>
<td>33.82 / 0.9226</td>
<td>29.76 / 0.8311</td>
<td>28.80 / 0.7963</td>
<td>27.15 / 0.8276</td>
<td>32.24 / 0.9343</td>
<td></td>
</tr>
<tr>
<td>LapSRN [13]</td>
<td>502K</td>
<td>33.81 / 0.9220</td>
<td>29.79 / 0.8325</td>
<td>28.82 / 0.7980</td>
<td>27.07 / 0.8275</td>
<td>32.21 / 0.9350</td>
<td></td>
</tr>
<tr>
<td>DRRN [20]</td>
<td>298K</td>
<td>34.03 / 0.9244</td>
<td>29.96 / 0.8349</td>
<td>28.95 / 0.8004</td>
<td>27.53 / 0.8378</td>
<td>32.71 / 0.9379</td>
<td></td>
</tr>
<tr>
<td>MemNet [21]</td>
<td>-</td>
<td>34.09 / 0.9248</td>
<td>30.00 / 0.8350</td>
<td>28.96 / 0.8001</td>
<td>27.56 / 0.8376</td>
<td>32.51 / 0.9369</td>
<td></td>
</tr>
<tr>
<td>IDN [10]</td>
<td>553K</td>
<td>34.11 / 0.9253</td>
<td>29.99 / 0.8354</td>
<td>28.95 / 0.8013</td>
<td>27.42 / 0.8359</td>
<td>32.71 / 0.9381</td>
<td></td>
</tr>
<tr>
<td>EDSR-baseline [16]</td>
<td>1,555K</td>
<td>34.37 / 0.9270</td>
<td>30.28 / 0.8417</td>
<td>29.09 / 0.8052</td>
<td>28.15 / 0.8527</td>
<td>34.35 / 0.9493</td>
<td></td>
</tr>
<tr>
<td>SRMDNF [24]</td>
<td>1,526K</td>
<td>34.12 / 0.9254</td>
<td>30.04 / 0.8362</td>
<td>28.97 / 0.8025</td>
<td>27.57 / 0.8398</td>
<td>33.00 / 0.9403</td>
<td></td>
</tr>
<tr>
<td>CARN [2]</td>
<td>1,592K</td>
<td>34.29 / 0.9255</td>
<td>30.29 / 0.8407</td>
<td>29.06 / 0.8034</td>
<td>28.06 / 0.8493</td>
<td>33.50 / 0.9440</td>
<td></td>
</tr>
<tr>
<td>IMDN [9]</td>
<td>703K</td>
<td>34.36 / 0.9270</td>
<td>30.32 / 0.8417</td>
<td>29.09 / 0.8046</td>
<td>28.07 / 0.8519</td>
<td>33.61 / 0.9445</td>
<td></td>
</tr>
<tr>
<td>RFDN-L [17]</td>
<td>633K</td>
<td>34.47 / 0.9280</td>
<td>30.35 / 0.8421</td>
<td>29.11 / 0.8053</td>
<td>28.32 / 0.8547</td>
<td>33.74 / 0.9458</td>
<td></td>
</tr>
<tr>
<td>MAFFSRN [19]</td>
<td>807K</td>
<td>34.45 / 0.9277</td>
<td>30.40 / 0.8432</td>
<td>29.13 / 0.8061</td>
<td>28.26 / 0.8552</td>
<td>32.87 / -</td>
<td></td>
</tr>
<tr>
<td>LatticeNet [18]</td>
<td>765K</td>
<td>34.53 / 0.9281</td>
<td>30.39 / 0.8424</td>
<td>29.15 / 0.8059</td>
<td>28.33 / 0.8538</td>
<td>32.43 / -</td>
<td></td>
</tr>
<tr>
<td>ESRT (ours)</td>
<td>707K</td>
<td>34.42 / 0.9268</td>
<td>30.43 / 0.8433</td>
<td>29.15 / 0.8063</td>
<td>28.46 / 0.8574</td>
<td>33.95 / 0.9455</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Quantitative comparison with SISR models. The Best and the second-best results are highlighted and underlined, respectively.
Figure 1. Visual comparison with lightweight SISR models. Obviously, ESRT can reconstruct realistic SR images with sharper edges.
ARFBs | Set5 | Set14 | BSD100 | Urban100 | Manga109
---|---|---|---|---|---
1 | 32.17dB | 28.59dB | 27.55dB | 26.23dB | 30.61dB
2 | 32.20dB | 28.69dB | 27.70dB | 26.37dB | 30.74dB

Table 2. Study on the impact of different numbers of ARFB in the lower branch on model performance (x4).

ARFBs | Set5 | Set14 | BSD100 | Urban100 | Manga109
---|---|---|---|---|---
0 | 32.16dB | 28.59dB | 27.53dB | 26.22dB | 30.61dB
1 | 32.21dB | 28.69dB | 27.69dB | 26.39dB | 30.75dB
2 | 32.20dB | 28.71dB | 27.71dB | 26.44dB | 30.78dB

Table 3. Study on the impact of different numbers of ARFB in the upper branch on model performance (x4).

Adaptive | Set5 | Set14 | BSD100 | Urban100 | Manga109
---|---|---|---|---|---
X | 32.13dB | 28.58dB | 27.56dB | 26.24dB | 30.62dB
✓ | 32.19dB | 28.69dB | 27.69dB | 26.39dB | 30.75dB

Table 4. Study on the impact of adaptive scaling on model performance (x4).

Case | PSNR(dB) | Param.(K) | GPU Memory
---|---|---|---
w/o TR | 31.96 | 554 | 1931M
Original TR [22] | 32.14 | 971 | 16057M
1 ET | 32.18 | 751 | 4191M
2 ET | 32.25 | 949 | 3159M
s=1 | 32.14 | 751 | 41580M
s=2 | 32.15 | 751 | 6731M
s=4 | 32.18 | 751 | 4191M
s=6 | 32.04 | 751 | 3159M

Table 5. Study of Efficient Transformer (ET) on Set5 (x4). The GPU memory here refers to the cost of the model during training, which patch size = 48*48 and batch size = 16.

### 1.3. Study of Adaptive Scaling

ARFB contains two Residual Units (RUs) and two convolutional layers, which is one of the most basic components to build the High Preserving Block (HPB). Meanwhile, a residual scaling with adaptive weights (RSA) is designed to dynamically adjust the importance of residual path and identity path. To verify the effectiveness of the adaptive scaling mechanism, we provide an ablation study in Table 4. Among them, X and ✓ represent the models with and without the adaptive scaling mechanism in ARFB, respectively. Obviously, with the help of the adaptive scaling mechanism, the performance of the model can be further improved. This fully demonstrates the effectiveness of the adaptive scaling mechanism.

### 1.3.1 Study of Efficient Transformer (ET)

To capture the long-term dependencies of similar local regions in the image, we introduced the Transformer and proposed a Efficient Transformer (ET). To illustrate the efficiency and effectiveness of ET, we provide the following experiments:

**A. TR v.s. w/o TR:** Firstly, we analyze the model with and without Transformer in Table 5. We can see that if we remove the Transformer, the model performance descends obviously from 32.18dB to 31.96dB. From this case, it can be inferred that the correlation of long-term image patches is beneficial for image super-resolution. The reason is that a natural scene image has many similar pixel blocks and these blocks always can complete other missing information as a reference. Therefore, the introduced Transformer can make full advantage of this relationship.

**B. ET v.s. Original TR:** Secondly, we compare our ET with the original Transformer in computer vision (ViT [8]). From Table 5 we can see that for the original TR, it will increase 417M parameters while our ET (1 ET) only increases 197M parameters. This benefits from the Reduction module that can reduce the number of channels. In addition, for GPU memory, the original TR occupies 16057M memory which even cannot run on some common NVIDIA GPUs like 1080Ti and 2080Ti. Contrastly, our ET just occupies 4191M GPU memory, which is only 1/4 of the original Transformer. More surprising is that the performance of the model with the original Transformer is even worse than our ESRT (1 ET). This is because the model with the original Transformer needs more data to train while the datasets are usually small in the SISR task. This experiment further verified the effectiveness of our proposed ET.

**C. The Number of ET:** In general, increasing the number of convolutional layers can increase the model performance. In view of this, we also added the number of ET in our model to explore its performance. From Table 5, we can see that when the number of ET increases, the model performance will be further improved. However, it is worth noting that the model parameters and GPU memory will also increase when the number of ET increases. Therefore, to keep consistent with other lightweight models in the aspect of parameters, only one ET is used in the final ESRT.

**D. The Splitting Factor s:** In MHA, a Feature Split Module (FSM) is used to split the original Q, K, and V into s segments to save the GPU memory. Commonly, the s is larger, the split segments are shorter and the GPU memory occupation is less. In Table 5, we investigate the different value of s. Obviously, the model achieves the best performance when s = 4. Meanwhile, we can observe that the change of s will not affect the number of model parameters. Therefore, we set s = 4 in the final model.
for the lack of feature extraction capabilities of the Trans-ESRT. This means that LCB can effectively compensate of Pure ESRT will significantly decrease compared with is set to BLE modified Transformer as “Pure ESRT”. According to TA-

Study of the Pure Transformer

In general, the pure Transformer-based architecture is more efficient and scalable than previous CNN-based architecture in both model size and computational cost. However, we find that the hybrid Transformer can perform better than the pure Transformer model on a lightweight model. To verify this view, we provide the performance of pure Transformer-based ESRT. Specifically, we modify ESRT to the pure Transformer by removing the LCB. We define the modified Transformer as "Pure ESRT". According to TABLE 6, it can be seen that if the number of ET in LTB is set to 1 in both Pure ESRT and ESRT, the performance of Pure ESRT will significantly decrease compared with ESRT. This means that LCB can effectively compensate for the lack of feature extraction capabilities of the Transformer. When the number of ETs in pure Transformer is increased to 3, the parameters of the model are close to our ESRT, but its performance is not as good as our ESRT and it will take up more GPU memory. This fully demonstrates the effectiveness of our proposed hybrid Transformer.

Meanwhile, we can see that our "Pure ESRT (8ET)" achieves close performance compared with the state-of-the-art method SAN [5] with only one-ninth parameters. Moreover, our model even achieves better performance on Urban100 than SAN. This reflects that building Pure-ESRT can achieve comparable SR performance compared with a well-designed CNN model.

1.5. Real Image Super-Resolution

In this part, we compare our ESRT with more classic lightweight SR models (e.g., SRCNN [6], VDSR [11], SR-ResNet [14], IMDN [9] and LK-KPN [3]) on the real image dataset (RealSR [3]). It is worth noting that since the resolution of the LR and HR images is the same in RealSR, the PixelShuffle is removed in our model and only one convolutional layer is applied to change the feature map into SR images. According to TABLE 7, we can observe that compared to IMDN, the performance of ESRT gains 0.07dB, 0.09dB, and 0.10dB for scaling factors ×2, ×3, and ×4, respectively. Also, our model has a close performance to LK-KPN which was specifically designed for the RealSR task. In addition, we provide the reconstructed SR images in Figure 2. Obviously, our ESRT recovers line edges effectively, such as some Chinese words and English words. Meanwhile, our ESRT can restore the texture details well, such as the grid lines in the air conditioner. All these ex-

<table>
<thead>
<tr>
<th>Model</th>
<th>Parameter</th>
<th>GPU occupy</th>
<th>Set5 PSNR/SSIM</th>
<th>Set14 PSNR/SSIM</th>
<th>BSD100 PSNR/SSIM</th>
<th>Urban100 PSNR/SSIM</th>
<th>Manga109 PSNR/SSIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESRT (Ours)</td>
<td>751K</td>
<td>4191M</td>
<td>32.13/0.8841</td>
<td>28.69/0.7833</td>
<td>26.50/0.7277</td>
<td>26.30/0.7902</td>
<td>30.15/0.9100</td>
</tr>
<tr>
<td>Pure ESRT (1ET)</td>
<td>357K</td>
<td>3967M</td>
<td>31.01/0.8751</td>
<td>27.85/0.7636</td>
<td>27.10/0.7203</td>
<td>25.00/0.7459</td>
<td>28.22/0.8726</td>
</tr>
<tr>
<td>Pure ESRT (2ET)</td>
<td>564K</td>
<td>5685M</td>
<td>31.77/0.8878</td>
<td>28.39/0.7758</td>
<td>27.42/0.7312</td>
<td>25.73/0.7728</td>
<td>29.76/0.8978</td>
</tr>
<tr>
<td>Pure ESRT (3ET)</td>
<td>771K</td>
<td>7409M</td>
<td>32.10/0.8926</td>
<td>28.59/0.7808</td>
<td>27.57/0.7360</td>
<td>26.13/0.7853</td>
<td>30.32/0.9057</td>
</tr>
<tr>
<td>Pure ESRT (4ET)</td>
<td>978K</td>
<td>9121M</td>
<td>32.29/0.8948</td>
<td>28.71/0.7830</td>
<td>27.64/0.7384</td>
<td>26.42/0.7936</td>
<td>30.69/0.9109</td>
</tr>
<tr>
<td>Pure ESRT (6ET)</td>
<td>1392K</td>
<td>12647M</td>
<td>32.36/0.8965</td>
<td>28.80/0.7850</td>
<td>27.70/0.7405</td>
<td>26.69/0.8016</td>
<td>30.97/0.9135</td>
</tr>
<tr>
<td>Pure ESRT (8ET)</td>
<td>1806K</td>
<td>16163M</td>
<td>32.40/0.8751</td>
<td>28.84/0.7858</td>
<td>27.73/0.7412</td>
<td>26.83/0.8048</td>
<td>31.11/0.9146</td>
</tr>
<tr>
<td>SAN [5]</td>
<td>15700K</td>
<td>1806K</td>
<td>32.64/0.9003</td>
<td>28.92/0.7888</td>
<td>27.78/0.7436</td>
<td>26.79/0.8068</td>
<td>31.18/0.9169</td>
</tr>
</tbody>
</table>

Table 6. Quantitative comparison between our ESRT, Pure ESRT, and SAN [5] (×4). The GPU memory here refers to the cost of the model during training, which patch_size = 48×48 and batch_size=16.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>×2</td>
<td>32.61/0.907</td>
<td>33.40/0.916</td>
<td>33.64/0.917</td>
<td>33.69/0.919</td>
<td>33.85/0.923</td>
<td>-</td>
<td>33.92/0.924</td>
</tr>
<tr>
<td>×3</td>
<td>29.34/0.841</td>
<td>29.96/0.845</td>
<td>30.14/0.856</td>
<td>30.18/0.859</td>
<td>30.29/0.857</td>
<td>30.60/0.863</td>
<td>30.38/0.857</td>
</tr>
<tr>
<td>×4</td>
<td>27.99/0.806</td>
<td>28.44/0.801</td>
<td>28.63/0.821</td>
<td>28.67/0.824</td>
<td>28.68/0.815</td>
<td>28.65/0.820</td>
<td>28.78/0.815</td>
</tr>
</tbody>
</table>

Table 7. PSNR and SSIM comparison with other advanced SISR methods on the RealSR dataset.

Figure 2. Visual comparison on RealSR dataset (version3). The SR images reconstructed by our ESRT have more accurate edges.
experiments show that our ESRT can also obtain a good SR property in the real world.

1.6. Different from other Transformer-base Methods

In recent years, some Transformer-base methods have been proposed for low-level image processing tasks. For example, In IPT [4], a novel Transformer-based network is proposed as the pre-trained model for low-level image restoration tasks. However, IPT utilizes ImageNet (more than 1.3M images) for training and has a huge number of parameters (115.5M), which is difficult to use in practical applications. In [23], RefSR uses different patches as the Q, K, and V to fuse various information of the reference images. Our method also follows this idea but the difference is that similar patches of ESRT are explored in the original image itself. Meanwhile, The complexity and computational cost of ESRT are lower than RefSR. In [15], a SwinIR is proposed for image restoration. The EMHA in our ESRT is similar to the Swin Transformer layer of SwinIR. However, SwinIR uses a sliding window to solve the high computation problem of the Transformer while ESRT uses a splitting factor to reduce the GPU memory consumption.

In addition, we selected the most representative SwinIR for comparison. We use the official code and test it on our server with the same environment and setting. The results are provided in Table 8. Obviously, our ESRT achieves close performance to SwinIR with fewer parameters and GPU memory. It is worth noting that SwinIR uses an extra dataset (Flickr2K [1]) for training, which is the key to further improving the model performance. For a fair comparison with methods such as IMDN, we did not use this external dataset in this work. All these results further validate the effectiveness of the proposed ESRT.

2. Discussions

Benefits of LCB. LCB solves the problem of the poor feature extraction ability of Transformer on small datasets. It is a lightweight architecture that can efficiently extract deep SR features. Meanwhile, LCB can be easily embedded into any SISR model to reduce model parameters and calculation costs, and maintain good performance.

Benefits of LTB. LTB solves the problem of heavy GPU memory consumption in vision Transformer. Meanwhile, ET can model the dependence between long-term sub-image blocks in the LR, enhancing the structural information of every image region. It has been improved that model such a long-term dependency of similar local regions is helpful for SR task. Meanwhile, ET is a lightweight and universal module that can be embedded into any present SR model to further improve model performance.

Limitations of ESRT. In this work, we propose a hybrid architecture consisting of CNN and Transformer. In order to keep the low complexity of the model, we directly connect the Transformer after the CNN. Although our experiments have verified the effectiveness of this method, we believe that there are more effective methods that can better utilize the local features extracted by CNN and the global relationship learned by Transformer. In future works, we will explore more effective combining methods to further improve the performance of the model.

Table 8. A detailed comparison of SwinIR and our ESRT (×4). The GPU memory here refers to the cost of the model during training, which patch size = 48×48 and batch size=16.

<table>
<thead>
<tr>
<th>Method</th>
<th>Dataset</th>
<th>Param.</th>
<th>GPU Memory</th>
<th>Set5</th>
<th>Set4</th>
<th>BSD100</th>
<th>Urban00</th>
<th>Manga109</th>
</tr>
</thead>
<tbody>
<tr>
<td>SwinIR</td>
<td>DIV2k + Flickr2K</td>
<td>897K</td>
<td>6966M</td>
<td>32.440/8976</td>
<td>28.770/7858</td>
<td>27.690/3406</td>
<td>26.470/7980</td>
<td>30.920/9151</td>
</tr>
<tr>
<td>ESRT</td>
<td>DIV2k</td>
<td>751K</td>
<td>4191M</td>
<td>32.190/8947</td>
<td>28.690/7833</td>
<td>27.690/3379</td>
<td>26.390/7962</td>
<td>30.750/9100</td>
</tr>
</tbody>
</table>

References


