LOW COMPLEXITY SINGLE IMAGE SUPER-RESOLUTION WITH CHANNEL SPLITTING AND FUSION NETWORK

Minqiang Zou, Jie Tang, Gangshan Wu

State Key Laboratory for Novel Software Technology, Nanjing University, China

ABSTRACT

Recently, deep convolutional neural networks (CNNs) have made remarkable progress on single image super-resolution (SISR). However, many of these methods use very deep or wide convolutional layers to achieve good performance, which treat all feature channels indiscriminately and neglect the difference among the contribution of each channel to the output results. In this paper, we propose a low complexity solution based on channel splitting and fusion network (CSFN) to address this problem. Our method uses channel splitting and channel fusion to enhance feature maps and make full use of valuable information, and then multiple residual channel splitting and fusion blocks (CSFB) are cascaded to continuously extract more important information for reconstruction. To further minimize redundant parameters and improve efficiency, we adopt group and recursive convolutional layer strategy in CSFB. Experiments demonstrate that our proposed CSFN could achieve higher performance with low computational complexity than most state-of-the-art methods.

Index Terms— Image super-resolution, convolutional neural networks, deep learning

1. INTRODUCTION

Single image super-resolution (SISR) is a classical computer vision task that generates a high-resolution (HR) image from a low-resolution (LR) one. Generally, SISR is regarded as an ill-posed problem since there are numerous HR images corresponding to one LR image. Several varieties of SR methods have been proposed to solve this inverse problem, such as interpolation-based methods, prediction-based methods [1, 2, 3], and example-based methods [4, 5, 6, 7]. In recent years, with the rapid development of computer performance and deep learning techniques, using deep learning to handle SR task is becoming a hot topic. These data-driven approaches have been proven to be able to achieve better results both in reconstruction accuracy and perceptual quality [8, 9].

Many previous works have proposed different efficient network structures for SR problem. Dong et al. [7] first propose a 3-layer CNN model named SRCNN, which has a lightweight structure and superior accuracy compared with the conventional example-based method. As stacking convolutional layers can increase the expressive ability of the network, Kim et al. [10] propose a very deep CNN model called VDSR. VDSR uses an extremely high learning rate and residual learning to ease the training difficulty in the deep model. FSRCNN [11] upsamples images using a deconvolution layer at the end of the network, which is much faster than the pre-upsampling method since it performs most of the mappings in low-dimensional space. ESPCN [12] uses sub-pixel convolution to learn the upsampling filters between LR and HR images, which is much faster than deconvolution. To encourage feature reuse and better propagate information, ResNet [13] and DenseNet [14] architecture are also widely used in SR tasks[15, 16, 17, 18], those methods can get superior performance by using deeper or wider networks.

Although a deeper network can get better results, it is hard to apply this huge model to practical applications due to the heavy computational and memory costs. To control the number of network parameters, recursive convolutional layers are used in DRCN [19], DRRN [20]. Usually, a recursive network would increase the depth of the model. Thus excessive recursive operations that saving intermediate results would lead to running out of the GPU memory when processing...
large images such as the case in DRRN[20], which is not suitable for real-world scenarios. CARN [21] is an accurate and lightweight deep network using the cascading mechanism to extract features. Chu et al. [22] propose a neural architecture search technology to produce a lightweight model, but this search technology is also based on a specific network structure to reduce searching space.

However, most CNN-based methods treat all feature channels indiscriminately and neglect the different contributions of each channel to the output results. Hu et al. [23] adopt a squeeze-and-excitation (SE) block to explicitly modeling the interdependencies between the channels of its convolutional features. RCAN [16] uses this channel attention mechanism in SR task to enhance discriminative learning ability, but all feature maps would flow into the next layer without distinction, which cannot make full use of useful channel features. To handle this inefficiency, we propose a new structure to consider the channel unevenness. In specific, we propose a deep residual channel splitting and fusion network (CSFN), which splits the channels to guide some features to generate more features in order to enhance feature maps gradually and make full use of the valuable information. In summary, our contributions can be summarized as follows: 1) We propose an efficient network, CSFN, for SISR to focus on channel information and different information fusion, which achieves high performance with low complexity. 2) To further reduce parameters and calculations, we design a new recursive CNN model (CSFN-M) to minimize redundant parameters. 3) Extensive experiments demonstrate that our CSFB can extract information efficiently and treat channels unequally, and after fusing that information, CSFB can provide more efficient feature maps for reconstruction.

2. PROPOSED METHOD

2.1. Network Architecture

As shown in Fig.2, our CSFN is divided into three parts: shallow feature extraction block (FEBlock), residual channel splitting and fusion blocks (CSFBlocks) and upscale block (UPBlock). Denote by $x$ and $y$ the input and output of CSFN. In FEBlock, we use one convolutional layer to extract features from the LR image. $F_0$ is the shallow features extracted by FEBlock. CSFBlocks consist of multiple CSFBs and a bottom convolution layer, and the output of CSFBlocks can be expressed as

$$F_r = f_1(H_{CSF,n}(\ldots (H_{CSF,1}(F_0) \ldots))) + F_0$$

where $f_1$ is the bottom convolution function and $H_{CSF,i}$ represents the operation of $i$-th CSFB. The detail of $H_{CSF,i}$ can be found in Section 2.2. The final part is UPBlock, which does the LR-HR transforming and reconstruct the HR image. We use ESPCN in our upscale module. For scale 2 and scale 3, we use one Conv-PixelShuffle structure. Two Conv-PixelShuffle modules are used for scale 4.

2.2. Residual Channel Splitting and Fusion Block

Our proposed CSFB (shown in Fig.3) can be roughly divided into two pipelines. The left pipeline is a feature fusion module based on channel split (CSFF) and the right one is a global feature extraction (GFE) module. Let’s denote the input feature maps of $i$-th CSFB as $F_{i-1}$, and the total number of channels in $F_{i-1}$ is $c_0$. Then the feature maps $F_{i-1}$ flow into CSFF, GFE and the other part of CSFB for residual learning.

CSFF module mainly focuses on the imbalance of channel information. Since the information extracted from LR images contains various information, the variance of channel features cannot be neglected. To make a distinction between different channel features, we adopt an asymmetric channel split to divide the feature into two parts. As Fig.3 shows, we split $F_{i-1}$ into two parts which contain $s$ and $c_0 - s$ channels respectively, where $s$ is less than $c_0/2$ in our network. We will discuss the purpose of this asymmetric splitting in Section 3.1. We denote the feature maps after splitting operation as $F_{i,s1}$ and $F_{i,s2}$, and the dimensions of left and right branch in CSFF as $c_a$ and $c_b$, subjected to $c_a + c_b = c_0$, $c_a > s$ is the restriction on $c_a$ to guide the network to extract more information in $F_{i,s1}$, $c_b < s$ can be explained as less feature maps in $F_{i,s2}$ would be used in this block. The operations of CSFF can be formulated as:

$$F_{i,s1} = \tau(f(\tau(f(F_{i,s1}, s, c_a), c_a, c_b)))$$

(2)

$$F_{i,s2} = \tau(f(\tau(f(F_{i,s2}, s - s, c_b), c_b, c_b)))$$

(3)

where $F_{i,s1}, F_{i,s2}$ denote the outputs of two branches, and $\tau$ is the ReLU activation function, $f(F, c_a, c_b)$ is the convolutional operation with $c_a$ as input feature dimension and $c_b$
as output feature dimension. Then the two feature maps are fused by concatenation and the 1×1 convolution.

The main purpose of GFE module (the right pipeline) is to compensate for CSFF module since CSFF module only uses partial channel information independently which leads to incomplete global information. A channel compression and expansion unit is used to extract features and promote channel information fusion as well as reducing the number of parameters. GFE is composed of 2 Conv-ReLUs and 1 Conv layer. The feature maps dimension of i-th Conv-ReLU output are denoted as \(c_n, c_n, \) and \(c_0\) respectively (\(c_n < c_0\)). This operation can be described as follows:

\[
F_{i,L} = f( f( f( F_{i-1}, c_0, c_n), c_n, c_n), c_n, c_0)
\]

(4)

The feature maps from CSFF and GFE are then aggregated to get the fusing features and residual learning. Thus, the output of i-th CSFB can be formulated as:

\[
F_{i} = H_{CSFF,i}(F_{i-1}) + H_{GFE,i}(F_{i-1}) + F_{i-1}
\]

(5)

where \(H_{CSFF,i}, H_{GFE,i}\) denote the operation collections in i-th CSFF and GFE modules respectively.

To further reduce parameters and enhance feature maps, we propose a more lightweight network(CSFMN-M) which replaces CSFB with CSFB-M. As shown in Fig.3 (right), we firstly adopt group convolutional layer in CSFF and GFE, and then the feature maps which flow into CSFB-M would be enhanced for \(t\) times. This can be described as follows:

\[
F_{i} = R_{t}(R_{t}(...R_{t}(F_{i-1})...))
\]

(6)

where \(R_{t}\) denotes the operation collections in i-th CSFB-M modules, and there are \(t R_{t}\) in the above formula.

### 3. IMPLEMENTATION DETAILS

In our proposed networks, the kernel sizes in convolutional layers are all set to \(3 \times 3\) except the specified \(1 \times 1\) convolutional layers. The number of channels in \(F_{i}(i = 1, 2, ..., n)\) is set to 64. We use 10 CSFBs in our proposed CSFN and 5 for CSFMN-M. In each CSFB, \(c_0, c_{s}, c_{a}, c_{b}\) and \(c_n\) are set to 64, 16, 32, 32 and 16 respectively. For our proposed CSFB-M, we use the same setting with CSFB but the number of groups is set to 2, \(t\) is set to 3 for each CSFB-M.

During the training process, for each batch, 16 LR RGB patches with the size of 48×48 are randomly selected from DIV2K dataset(800 training images). Then data augmentation is applied by flipping horizontally or vertically or rotating \(90^\circ\). The proposed network is implemented with PyTorch and optimized using Adam with \(\beta_1 = 0.9, \beta_2 = 0.9999\), and \(\epsilon = 10^{-8}\). The initial learning rate is set to \(10^{-4}\) and decreases by a factor of 2 for every \(2 \times 10^5\) iterations during the total \(10^6\) iterations. The loss function we used is \(L1\) loss. We use Set5, Set14, BSD100, and Urban100 for test and benchmark. The degradation operator we used is bicubic downsampling and the scaling factors are set to \(\times 2, \times 3, \) and \(\times 4\). We use two commonly used evaluation metrics in SR task to compare the rebuild quality: peak signal-to-noise ratio (PSNR) and structural similarity (SSIM).

### 3. EXPERIMENTS

#### 3.1. Network Analysis

**The effect of CSFB.** To investigate the effect of asymmetric splitting, we train our model roughly with different \(s\) in CSFF for total \(4 \times 10^5\) iterations. Specifically, we set the values of \(s\) to 8, 16, 24, and 32 respectively, and 5 CSFBs are used in this experiments, thus the four models have the same number of parameters, other settings are the same with Section 2.3. Fig.4 shows that appropriate asymmetric splitting can generate 0.07dB improvements compare with the symmetric splitting. We then adopt the ablation study to investigate the effects of CSFF and GFE in our CSFN. For better comparison and minimize the performance impact of rising parameter quantities, we only use 3 CSFBs to build a very shallow network. A ReLU-Conv module is used to replace the block in CSFB when there are no CSFF and GFE blocks in the network. Table 1 presents CSFF has slight improvement since it can not get global information from channels, GFE has worse performance with fewer parameters. After combining the CSFF and GFE, the performance would increase a lot compared with only using CSFF or GFE.

Since CSFF in our proposed CSFB plays a key role in the whole model, we then inspect the weights of the last 1x1 Conv layer in CSFF to explore the channel fusion of the two branches. The model we used is a trained \(\times 2\) model. Note that \(c_0 = 64\) and \(c_a = c_b = 32\), that is one output channel in CSFF is generated by left part of CSFF, \((c_a)\) and right part of CSFF, \((c_b)\), we then calculate the average weight (absolute value) of CSFF-L and CSFF-R and compute the weight ratio of CSFF-L to all channels. Fig. 5 shows that the first 16 channels use CSFF-L features more in all blocks because those feature maps would be used in the next CSFF-L, thus some feature map channels are constantly being strengthened during this process by using those features to build more features. Fig. 6 is the average (absolute value) of weights to build more features.
Table 2. Average PSNR/SSIMs for scale ×2, ×3 and ×4. Red/blue text: best/second-best, underline text: best result below 500K parameters.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Params/FLOPs</td>
<td>297K/802.9G</td>
<td>412K/17.7G</td>
<td>25.6K/177G</td>
<td>26K/177G</td>
<td>551K/177G</td>
<td>551K/177G</td>
<td>551K/177G</td>
<td>551K/177G</td>
<td>551K/177G</td>
<td>551K/177G</td>
</tr>
<tr>
<td>x2</td>
<td>Set5</td>
<td>31.27/0.916</td>
<td>33.26/0.914</td>
<td>33.35/0.915</td>
<td>33.32/0.915</td>
<td>33.30/0.915</td>
<td>33.30/0.915</td>
<td>33.30/0.915</td>
<td>33.30/0.915</td>
<td>33.30/0.915</td>
</tr>
<tr>
<td></td>
<td>BSD100</td>
<td>32.05/0.907</td>
<td>31.92/0.906</td>
<td>32.00/0.907</td>
<td>32.01/0.908</td>
<td>31.90/0.907</td>
<td>32.00/0.908</td>
<td>32.00/0.908</td>
<td>32.00/0.908</td>
<td>32.00/0.908</td>
</tr>
<tr>
<td></td>
<td>Urban100</td>
<td>31.23/0.918</td>
<td>31.23/0.918</td>
<td>31.23/0.918</td>
<td>31.23/0.918</td>
<td>31.23/0.918</td>
<td>31.23/0.918</td>
<td>31.23/0.918</td>
<td>31.23/0.918</td>
<td>31.23/0.918</td>
</tr>
<tr>
<td>x3</td>
<td>Set5</td>
<td>34.03/0.924</td>
<td>34.03/0.924</td>
<td>34.03/0.924</td>
<td>34.03/0.924</td>
<td>34.03/0.924</td>
<td>34.03/0.924</td>
<td>34.03/0.924</td>
<td>34.03/0.924</td>
<td>34.03/0.924</td>
</tr>
<tr>
<td></td>
<td>BSD100</td>
<td>32.96/0.849</td>
<td>30.00/0.849</td>
<td>30.10/0.849</td>
<td>30.13/0.849</td>
<td>30.13/0.849</td>
<td>30.13/0.849</td>
<td>30.13/0.849</td>
<td>30.13/0.849</td>
<td>30.13/0.849</td>
</tr>
<tr>
<td></td>
<td>Urban100</td>
<td>32.97/0.850</td>
<td>32.97/0.850</td>
<td>32.97/0.850</td>
<td>32.97/0.850</td>
<td>32.97/0.850</td>
<td>32.97/0.850</td>
<td>32.97/0.850</td>
<td>32.97/0.850</td>
<td>32.97/0.850</td>
</tr>
<tr>
<td>x4</td>
<td>Set5</td>
<td>34.30/0.925</td>
<td>34.30/0.925</td>
<td>34.30/0.925</td>
<td>34.30/0.925</td>
<td>34.30/0.925</td>
<td>34.30/0.925</td>
<td>34.30/0.925</td>
<td>34.30/0.925</td>
<td>34.30/0.925</td>
</tr>
<tr>
<td></td>
<td>BSD100</td>
<td>34.20/0.920</td>
<td>34.20/0.920</td>
<td>34.20/0.920</td>
<td>34.20/0.920</td>
<td>34.20/0.920</td>
<td>34.20/0.920</td>
<td>34.20/0.920</td>
<td>34.20/0.920</td>
<td>34.20/0.920</td>
</tr>
<tr>
<td></td>
<td>Urban100</td>
<td>34.20/0.920</td>
<td>34.20/0.920</td>
<td>34.20/0.920</td>
<td>34.20/0.920</td>
<td>34.20/0.920</td>
<td>34.20/0.920</td>
<td>34.20/0.920</td>
<td>34.20/0.920</td>
<td>34.20/0.920</td>
</tr>
</tbody>
</table>

Table 3. Performance of ×4 SR by CSFB and RCAB in terms of PSNR(dB). All models are trained in 4×10^5 iterations.

<table>
<thead>
<tr>
<th>Model</th>
<th>Params</th>
<th>Set5</th>
<th>Set14</th>
<th>BSD100</th>
<th>Urban100</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>742K</td>
<td>31.81</td>
<td>28.42</td>
<td>27.43</td>
<td>25.65</td>
<td>26.78</td>
</tr>
<tr>
<td>RCAB</td>
<td>745K</td>
<td>31.91</td>
<td>28.40</td>
<td>27.42</td>
<td>25.69</td>
<td>26.80</td>
</tr>
<tr>
<td>CSFB</td>
<td>743K</td>
<td>31.95</td>
<td>28.46</td>
<td>27.46</td>
<td>25.73</td>
<td>26.84</td>
</tr>
</tbody>
</table>

3.2. Comparison with State-of-the-Art Models

We compare our method with some state-of-the-art methods: VDSR [10], DRRN [20], MemNet [25], IDN [26], CARN [21] and SRFBN [24]. For a fair comparison, the heavy network such as EDSR, DBPN, RDN, RCAN is excluded. To better reveal the computational complexity of each model, we calculate the floating-point operations (FLOPs) under the assumption that the size of the output image is 1280×720. Table 2 shows our CSFN has better performance with fewer parameters than RCAB for SR tasks.

4. CONCLUSION

In this paper, we propose a novel low complexity residual channel splitting and fusing network (CSFN) architecture which takes into account the unevenness in feature map channels and can extract features more efficiently for SISR task. Our proposed method can reconstruct complex objects accurately and show superior results considering the PSNR and the SSIM. This network will be meaningful for practical application. In the future, this work will be used to image restoration and video super-resolution tasks.
5. REFERENCES


