Journal of Real-Time Image Processing (2019) 16:81–97
https://doi.org/10.1007/s11554-018-0774-z

SPECIAL ISSUE PAPER

A fast single-image super-resolution method implemented with CUDA

Yuan Yuan[1] · Xiaomin Yang[1] · Wei Wu[1] · Hu Li[1] · Yiguang Liu[2] · Kai Liu[3]

Received: 4 September 2017 / Accepted: 13 February 2018 / Published online: 10 April 2018

Abstract
Image super-resolution (SR) plays an important role in many areas as it promises to generate high-resolution (HR) images without upgrading image sensors. Many existing SR methods require a large external training set, which would consume a lot of memory. In addition, these methods are usually time-consuming when training model. Moreover, these methods need to retrain model once the magnification factor changes. To overcome these problems, we propose a method, which does not need an external training set by using self-similarity. Firstly, we rotate original low-resolution (LR) image with different angles to expand the training set. Second, multi-scale Difference of Gaussian filters are exploited to obtain multi-view feature maps. Multi-view feature maps could provide an accurate representation of images. Then, feature maps are divided into patches in parallel to build an internal training set. Finally, nonlocal means is applied to each LR patch from original LR image to infer HR patches. In order to accelerate the proposed method by exploiting the computation power of GPU, we implement the proposed method with compute unified device architecture (CUDA). Experimental results validate that the proposed method performs best among the compared methods in both terms of visual perception and objective quantitation. Moreover, the proposed method gets a remarkable speedup after implemented with CUDA.

Keywords Super-resolution · Self-similarity · GPU · CUDA

1 Introduction

High-resolution (HR) image plays an important role in many fields such as medical diagnosis, high-resolution displayer, security system, and so on. However, it is difficult to capture images with desired resolution because of the limitation of hardware. It is expensive and even impossible to upgrade the hardware in many conditions, for example, image sensors on satellite. Enlarging image with image processing methods has received extensive attention in recent years. The simplest method to enlarge image is interpolation operation. However, the interpolated HR image, which lacks high-frequency (HF) details, has obvious aliasing.

Super-resolution (SR) [1] is a promising modern image processing method to enhance image resolution from low-resolution (LR) image. Generally, SR methods can be divided into two categories, i.e., multi-frame-based (also known as reconstruction based) methods and single-frame-based (also known as learning based) methods. Multi-frame-based SR methods [2–5] reconstruct HR image using a sequence of LR images from the same scene. Multi-frame-based SR methods highly depend on the quality of the registration of different input images. In practice, it is hard to obtain images that satisfy the condition.

Single-frame-based image SR methods [6–12], which employs training set containing LR and HR pairs as prior information source, have become a top topic in recent years. Depending on the source of training set, these methods can be classified into two categories, i.e., external training set-based methods and internal training set-based methods. A large training set from other images is required in external-based methods. Yang et al. [9] proposed a method based on spare representation and dictionary learning. They used external LR and HR pairs to train LR

✉ Wei Wu
wuwei@scu.edu.cn

[1] College of Electronics and Information Engineering, Sichuan University, Chengdu 610064, Sichuan, China

[2] College of Computer Science, Sichuan University, Chengdu 610064, Sichuan, China

[3] College of Electrical and Engineering Information, Sichuan University, Chengdu 610064, Sichuan, China

and HR dictionary, respectively. Yang et al. [10] proposed a sparse coding-based SR method to enlarge infrared image. They exploited fuzzy clustering to learn multiple dictionaries. Wu et al. [11] proposed a multiple dictionaries-based framework to upscale remote images. They exploited multi-type features to present images. Dong et al. [12] proposed a deep learning method for single-frame SR. They used a deep convolutional neural network to learn the mapping relationship between the low-/high-resolution images. All the above-mentioned methods need a large-scale external training set, which would consume a lot of memory. A large-scale external training set can provide plenty of prior information. However, the number and types of training images required for satisfactory levels of performance are not clear [13]. Moreover, these methods need to retrain model once the magnification factor changes. In addition, training a model with satisfactory performance is usually time-consuming.

To avoid using external training datasets and their associated problems, image similarity and image self-similarity are explored by researchers. In [14], image similarity is used in the process of distinguishing similar images from image copies to find out illegal image copies. In [15], Li et al. proposed a novel method to detect copy-move forgery by using image similarity within an image. In [16], Pan et al. proposed a novel method for reducing the computational complexity of multi-view video coding with using image similarity. In [17], Rong et al. proposed a method to protect privacy within images in detected communities by exploiting image similarity. In [18], image similarity is used in the process of clustering images to eliminate near-duplicate images collected by visual sensor nodes. In [19], Zhou et al. proposed an effective image copying detection method which can detect arbitrary rotation copies of original image by exploiting image similarity.

Some SR methods [13, 20–22] exploit internal training set by using self-similarity of images. These methods take advantage of self-similarity of images, which means image patches recur within and across scale of the same image [13]. Zeyde et al. [8] proposed a sparse representation-based method, which could operate without an external training set by boot-strapping the scale-up task from the given LR image. Huang et al. [13] proposed a self-similarity-driven SR method. They built a compositional transformation model to generate transformed self-example, which expands the internal patch searching space. A method in our previous research [22] applied generalized nonlocal means to self-similarity-based SR method. Internal training set was built from down-sample version images, and HR details were estimated by using a method based on generalized nonlocal mean. All the previous mentioned methods using internal training set suffer from taking a long consuming time, which cannot meet the requirement of real-time and/or near-real-time applications.

With the programmability on the GPU becoming easier, the GPU has been exploited to accelerate SR methods recently. Sun et al. [23] accelerated a multi-frame-based SR method by using CUDA. The deep neural network-based SR method proposed by Dong et al. [12] exploited CUDA deep neural network (CUDNN) library to accelerate their training process.

Most internal training set-based SR methods have a good performance in both reconstruction quality and consuming time relatively. However, it suffered from the following limitations:

1. The internal training set, which was built from the down-sample version of input LR image in original direction, was not comprehensive enough [13, 20–22]. Some important HF details information could not be integrated into resultant images.
2. Only single-scale filter was employed to extract feature map from an image [13, 20–22]. Feature map extracted by single-scale filter failed to represent an image accurately, which further leads to artifacts in reconstructed images.
3. The computation power of GPU was not exploited to accelerate internal training set-based SR methods.

Inspired by the using of GPU in SR and based on our previous work, we proposed a method in this paper with the following three contributions:

1. In order to expand training set, we rotate the input LR image with four different angles. The expanded training set can provide plenty of prior information.
2. In order to represent an image more accurately, we exploit multi-scale DoG filters to extract feature maps from an image.
3. In order to meet the requirement of real-time and/or near-real-time applications, we implement our method with CUDA for high-performance computation.

The rest of this paper is organized as follows. Section 2 reviews the related work on self-similarity-based SR method and CUDA programming model. Details of the proposed method and the implementation details with CUDA are described in Sect. 3. Experimental results on four datasets are illustrated in Sect. 4. Section 5 summarizes the conclusions of this paper.

2 Related work

2.1 Self-similarity-based SR method

As we have mentioned in Sect. 1, images have the nature called self-similarity. Patches of a nature image recur within same and different scale of the same image. Figure 1 illustrates this phenomenon. As shown in Fig. 1, a patch marked with red box and another marked with green box are similar in each image. This phenomenon is self-similarity in same scale. The patch marked with red box can be matched with multiple similar patches in down-sample version image. This is self-similarity in different scale.

An internal training set containing LR and HR images (or patches) can be built by taking advantage of this nature. The input LR image can be regarded as an HR version image in training set, while its corresponding down-sample image can be regarded as an LR version image. Multiple LR/HR pairs built through different down-sample factors can be organized together as an integral training set. Figure 2 represents how to build internal training set.

The learning-based SR can be described as estimating the best HR image when the LR image is known. This can be expressed as:

$$I_H = \arg\max_{I_H} P(I_H|I_L) \tag{1}$$

where I_H and I_L denote HR image and LR image, respectively. $P(\cdot|\cdot)$ refers to conditional probability. An HR image can be decomposed into an LR image and an HF map, which can be expressed as:

$$I_H = H + I_L \tag{2}$$

where H denotes HF map. From Eq. (2), estimating I_H directly can be converted to estimate H indirectly. Suppose that $I_L = L + M$, where L and M denote low-frequency (LF) map and middle-frequency map (MF), respectively. According to [22], Eq. (1) can be equivalently rewritten as:

$$H = \arg\max_{H} P(M, H) \tag{3}$$

where $P(M, H)$ denote the joint probability of MF map and HF map. Thus, the MF map of LR image rather than LR image itself is used to estimate HF map indirectly. According to the theory in [22], joint probability, $P(M, H)$ can be equivalently written as:

$$\begin{aligned} P(M, H) &= P(M_1 M_2, \ldots, M_N, H_1 H_2, \ldots, H_N) \\ &= \prod_{n=1}^{N} P(M_n, H_n) \end{aligned} \tag{4}$$

where M_n is a patch from MF map of input LR image. H_n, which is unknown, is the patch we need to infer from HF map in training set. According to Eq. (4), maximizing $P(M, H)$ is equivalent to maximizing each $P(M_n, H_n), n = 1, 2, 3, \ldots, N$. $P(M_n, H_n)$ can be expressed as:

$$P(M_n, H_n) = \exp\left(-\frac{\left|M_n - M_n^H\right|^2}{2\sigma^2}\right) \tag{5}$$

where M_n^H is the MF patch corresponding to H_n. σ is a parameter. According to Eq. (5), maximizing $P(M_n, H_n)$ is equivalent to minimizing $\left|M_n - M_n^H\right|$. Thus, Eq. (3) can be equivalently written as:

$$H_n = \arg\min\left|M_n - M_n^H\right| \tag{6}$$

According to Eq. (6), the HF patch H_n, whose corresponding MF patch M_n^H is the most similar patch to the input MF patch M_n, is the best estimated HF patch. According to [22], final estimated $\widehat{H_n}$ can be formulated as:

$$\widehat{H_n} = \sum_{k}^{K} \omega_k H_k^T \tag{7}$$

where, $\widehat{H_n}$ is the estimated HF patch. H_k^T denotes the k-th similar HF patch in training set. $\omega_k, k = 1, 2, 3, \ldots, K$ refers to the weight coefficient of H_k^T. One of the cores of this method is to calculate ω_k. According to [22], ω_k can be calculated by Eq. (8).

$$\omega(i, j) = \frac{1}{Z(i)} \exp\left(-\frac{\left|M_i - M_j\right|^2}{2\sigma^2}\right) \tag{8}$$

where $Z(i)$ is a normalizing factor. It is calculated as:

$$Z(i) = \sum_{i} \exp\left(-\frac{\left|M_i - M_j\right|^2}{2\sigma^2}\right) \tag{9}$$

where σ is a parameter.

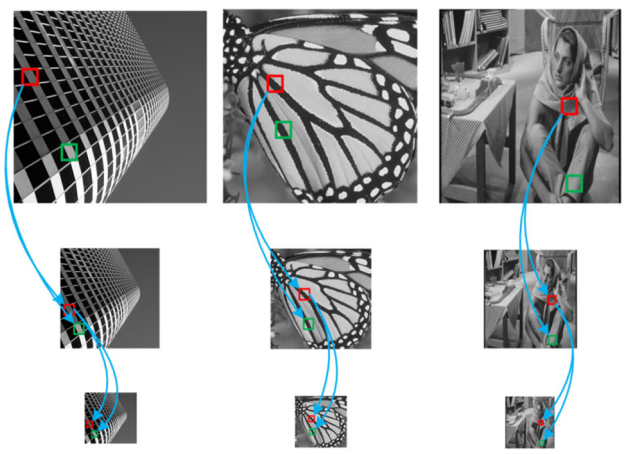


Fig. 1 Examples of self-similarity

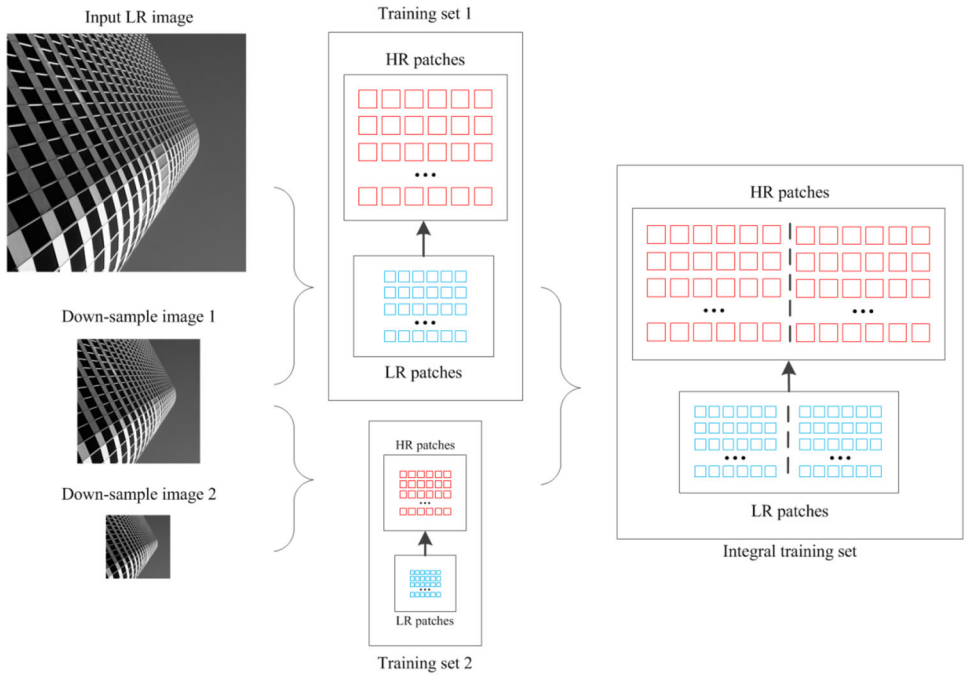


Fig. 2 Building an internal training set

2.2 CUDA programming model

Compute unified device architecture (CUDA), which is proposed by NVIDIA Corporation, is a GPU-based parallel computing architecture. It takes advantage of massive cores integrated on GPU chip to help users to execute algorithms in parallel. CUDA programming model contains heterogeneous programming, threads hierarchy and memory hierarchy.

Heterogeneous programming In CUDA programming model, GPU is defined as device, while CPU is defined as host. CPU and GPU work corporately to complete all operations including data I/O, data computation and interaction operation. Usually, CPU is responsible for tasks which contain complex logic controls and data operation with disks (defined as serial codes). Intensive computation, which can be divided into independent fine-grained tasks (defined as parallel kernel function), is assigned to GPU for parallel execution. Figure 3 presents the heterogeneous execution model.

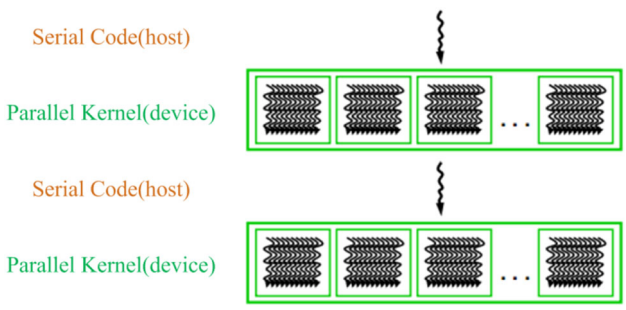


Fig. 3 Heterogeneous programming in CUDA

Threads hierarchy In terms of hardware, a CUDA device is a general purpose GPU (GPGPU) consisting of a set of multicore processors, defined as streaming multiprocessors (SMs). Streaming processors (SPs) within a SM work in a single-instruction multiple-data (SIMD) fashion. Hundreds or even thousands of cores are integrated on a GPU chip in this way. In terms of software, these CUDA threads (cores) are logically organized by a Grid-Block model: multiple threads form a block and multiple blocks form a grid. Thread, block and grid are assigned a unique 3-component vector comprised of one-, two- or three-dimensional index. Figure 4 illustrates this hierarchy. This organization provides users a flexible way to divide their algorithms into fine-grained tasks. For example, a voice signal which is treated as a 1D data can be processed conveniently with a 1D thread block, and it is a nature way to exploit a 2D thread block to process an image signal.

The parallel design in CUDA is based on the SIMT. There are three levels of parallel granularity in CUDA: thread, wrap and thread block [24]. A thread processes a single datum, such as an image pixel or an image patch, whose memory address is located by the ID of corresponding thread. A warp containing 32 threads is the fundamental unit of SIMD, that is to say instructions are dispatched to 32 threads within a wrap simultaneously. A set of wraps form a thread block. Each thread block executes on a SM and manages a pool of wraps. A set of SMs on GPU execute simultaneously.

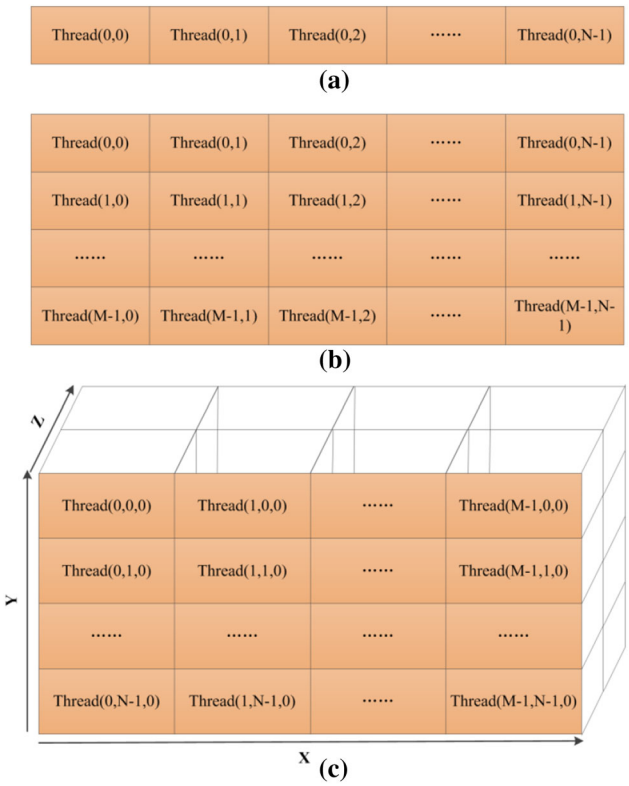


Fig. 4 Threads hierarchy in CUDA. **a**: A 1D block. **b** A 2D block. **c** A 3D block

Memory hierarchy There are five memory spaces in CUDA: global, texture, constant, shared, and register memory. Figure 5 illustrates these memory spaces [25].

- Global memory, which is off-chip memory with a large capacity, has a high latency. Usually, global memory plays a role as a bridge between CPU memory space and GPU memory space. Data are copied from CPU memory space to GPU global memory space firstly and copied back after procedure completed. In order to reduce access latency, global memory should be restricted to fully coalesced access.

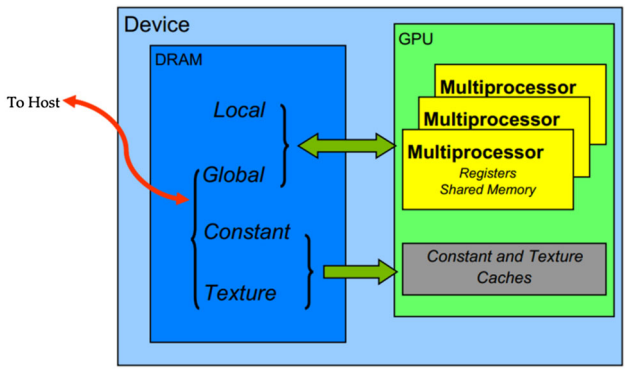


Fig. 5 CUDA memory space [25]

- Texture and constant memory are read-only and they have cache. The cache could lead to high-access performance. Constant memory can only store a small number of constants because of its limitation on capacity. Texture memory has plenty of capacity because it is bound to global memory.
- Shared memory and register memory are types of on-chip memory. Accessing shared memory is as fast as accessing registers, as long as no bank conflict exists [24]. Shared memory plays an important role in optimizing CUDA programs. Data with local property are usually copied from global memory to shared memory in a block firstly, and then all the threads within this block can access the shared data with a low latency. Each SM has a fixed number of registers. These registers are assigned equally to active threads within this SM. Excessive use of registers in a kernel limits the number of threads that can run simultaneously [24].

3 The proposed method

3.1 The framework of the proposed method

There are two phases in the proposed method: the training phase and the inferring phase. An internal training set was built by taking advantage of self-similarity of input LR image in the training phase. In the inferring phase, the HR image was obtained by adding the interpolated image and the estimated HF map, which was inferred from the training set by exploiting nonlocal means. Figure 6 presents the framework of the proposed method.

In our method, an HF map is obtained by differential operation between the LR image and its blurred image. The blurred image is obtained with two steps: (1) Gaussian blur and down-sample operation are exploited on input LR image. (2) The blurred image is obtained by upscaling the down-sample version image. The MF map is obtained by DoG filter. According to Sect. 2.1, an LR image can be decomposed into an LF map and a MF map; therefore, the MF map can be obtained by filtering the LF map. DoG filter is exploited in this paper to achieve that goal.

There are three innovative places in the proposed method: (1) the input LR image is rotated with four different angles to obtain four rotated images. The four rotated images are used to build the internal training set because these rotated images could expand the training set. (2) Multi-scale DoG filters are exploited to extract multi-view MF maps in both training phase and inferring phase. We organize these maps together to form concatenate feature vectors, which are used to represent an image.

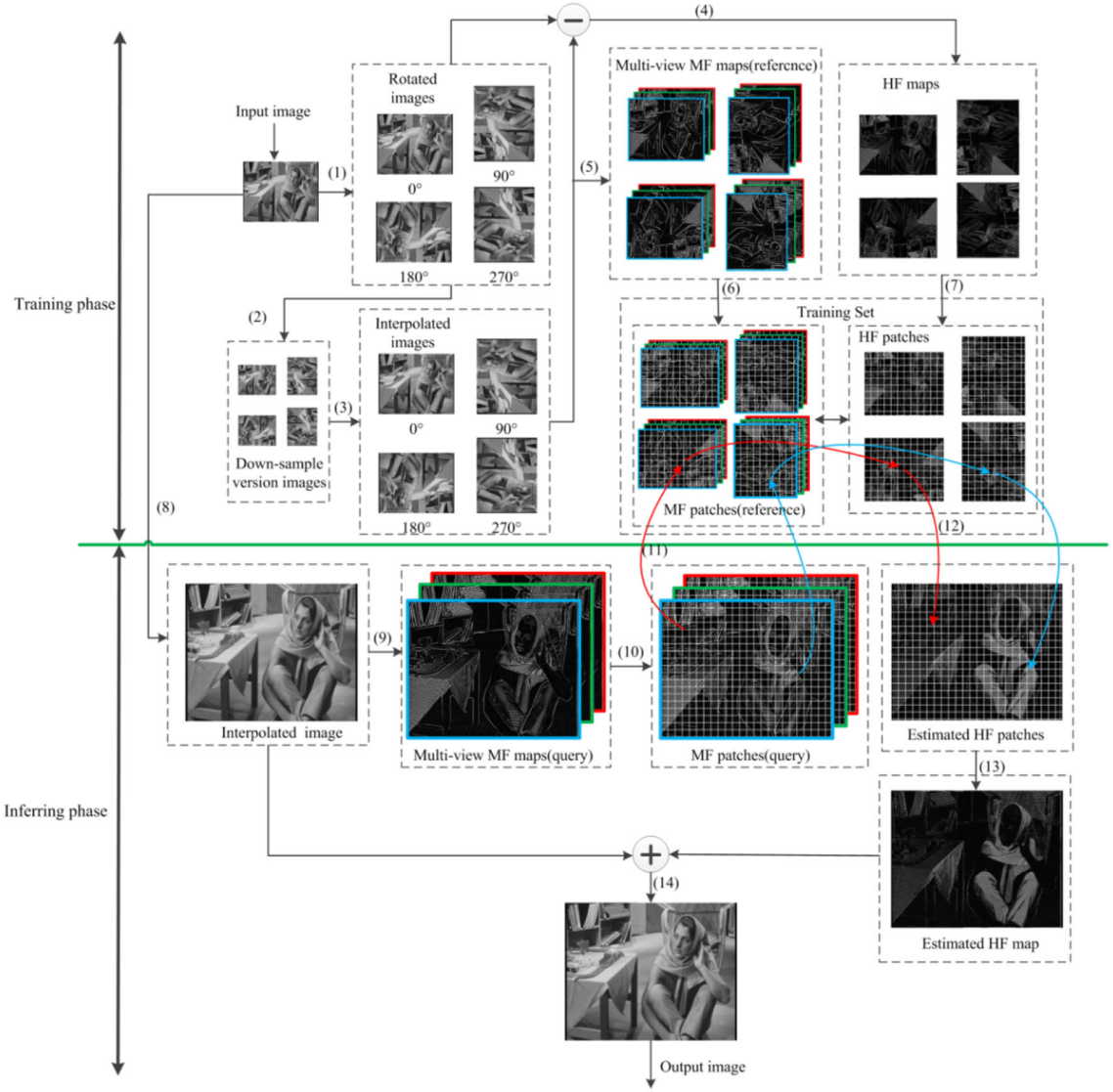


Fig. 6 The framework of the proposed method

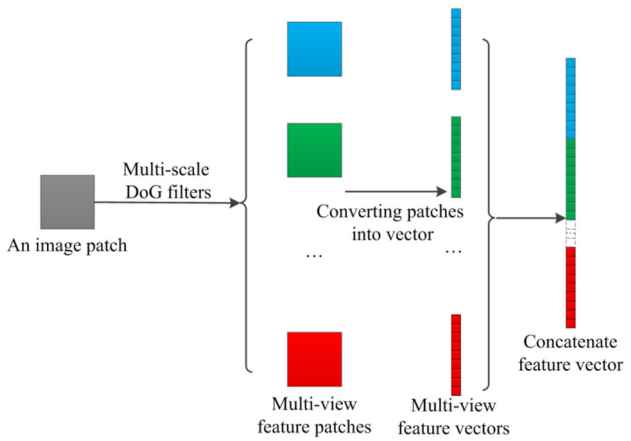


Fig. 7 The process of generating concatenate feature vectors

Figure 7 shows this process. Experimental results invalidate that concatenate feature vectors could represent an image more accurately. (3) We implement the proposed method with CUDA for high-execution effectiveness.

3.1.1 The training phase

The training phase consists of seven steps (1)–(7), as shown in Fig. 6. The details of the steps are as follows:

1. Firstly, the input LR image is rotated with 0°, 90°, 180°, and 270°. This operation can be formulated as follows:

$$I_L^r = Rot(I_L) \tag{10}$$

where I_L and $I_L^r, r \in \{0^\circ, 90^\circ, 180^\circ, 270^\circ\}$ denote the input LR image and the corresponding rotated image, respectively. $Rot(\cdot)$ refers to rotation operation.

2. Second, Gaussian blur and down-sample operation are applied to the rotated image I_L^r, which can be expressed as follows:

$$I_{LL}^r = \left(G * I_L^r\right) \downarrow \tag{11}$$

where G and * denote Gaussian blur kernel and convolution operation, respectively. $\downarrow$ refers to down-sample operation. I_{LL}^r is the down-sample version of I_L^r.

3. Then, the I_{LL}^r is enlarged to I_{LH}^r whose size is same as I_L^r by using interpolation. Bicubic-linear interpolation is used this paper. This process can be expressed as follows:

$$I_{LH}^r = Intp\left(I_{LL}^r\right) \tag{12}$$

where $Intp(\cdot)$ refers to interpolation operation.

4. The HF map can be obtained by applying differential operation on I_{LH}^r and I_L^r, which can be formulated as follows:

$$H^r = I_L^r - I_{LH}^r \tag{13}$$

5. Multi-scale DoG filters are applied to I_{LH}^r to extract MF map defined as $M_d^r, d \in \{1, 2, \ldots, D\}$, which can be calculated by the following equation:

$$M_d^r = DoG\left(I_{LH}^r\right) \tag{14}$$

where $DoG(\cdot)$ refers to extract image feature with DoG filter. D denotes the number of DoG filters.

6. MF maps $M_d^r, d \in \{1, 2, \ldots, D\}$ are divided into concatenate feature vectors $V_{MR}^i, i = 1, 2, 3, \ldots, M$ to form searching space, which is also called reference space. M is the total numbers of reference feature vectors.
7. HF maps H^r, $r \in \{0^\circ, 90^\circ, 180^\circ, 270^\circ\}$ are divided into HF patches $P_{HR}^i, i = 1, 2, 3, \ldots, M$. P_{HR}^i and V_{MR}^i obtained at step (6) have a strict corresponding relationship. These HF patches, which is defined as reference HF patches, are corpus to estimate HF patches of output HR image.

3.1.2 The inferring phase

The inferring phase consists of seven steps (8)–(14), as shown in Fig. 6. The details of inferring phases are shown as follows:

8. The input LR image is enlarged by using bicubic-linear interpolation operation and the interpolated image is defined as I_{Intp}.
9. The same DoG filters mentioned in the training phase are applied to the interpolated image I_{Intp} to get MF maps, which can be expressed as follows:

$$M_{Intp}^d = DoG(I_{Intp}) \tag{15}$$

where M_{Intp}^d, $d \in \{1, 2, \ldots, D\}$ denotes the MF map of interpolated image I_{Intp}.

10. The MF map M_{Intp}^d is divided into concatenate feature vectors $V_{MQ}^i, i = 1, 2, 3, \ldots, N$ as query objects. N is the total number of query objects.
11. For a given query object V_{MQ}^i, we search for its K most similar patches in the reference space. Distances between the query object and its K most similar results are defined as D; the indexes of the K most similar results in reference space are defined as I. Euclid distance is exploited in this paper.
12. The estimated HF patches can be calculated by the following formulator:

$$P_{HE}^i = \sum_{k=1}^{K} \omega_k P_{HR}^{I(k,i)} \tag{16}$$

where ω_k is the k-th coefficient which can be calculated by Eq. (8).

13. All the estimated HF patches are merged together to form an estimated HF map.
14. The final output image is obtained by adding the interpolated image and the estimated HF map, which can be expressed as follows:

$$I_H = I_{Intp} + \widehat{H} \tag{17}$$

where $\widehat{H}$ and I_H denote estimated HF map and final output HR image, respectively.

3.2 Implementation details with CUDA

3.2.1 Accelerating SR method with CUDA

With CUDA, GPU has been recently exploited to accelerate computationally intensive tasks, such as deep learning, virtual reality, autonomous vehicles and so on [26]. There are two key points in accelerating an algorithm with CUDA: dividing original algorithm into fine-grained tasks and enhancing memory access to reduce access latency.

The same as other computationally intensive applications, image super-resolution can also be accelerated with

CUDA. There are two further principles why CUDA can be exploited on SR: many SR algorithms work on patch-wise which ensure fine-grained threads parallelism, and the image data have local property which means data parallelism can be achieved. Figure 8 presents how to achieve fine-grained threads parallelism and data parallelism for SR. In SR, an image is divided into patches and these patches are organized into groups according to specified group size. Each patch group is bound to a thread block in CUDA. Each bound thread block executes on a SM. The data within a patch group are copied to shared memory space from global memory space firstly, and then each thread within the bounded thread block processes its corresponding data in share memory space.

In most of learning-based SR algorithms, building training sets and inferring HF patches can be accelerated by using CUDA. However, some particular method cannot be parallelized easily, for example, the KSVD algorithm used in sparse representation-based methods, whose nested loop is not independent with each other.

3.2.2 Implementation details with CUDA

3.2.2.1 Implementation of the training phase The MF maps $M_d^r, d \in \{1, 2, \ldots, D\}$, $r \in \{0^\circ, 90^\circ, 180^\circ, 270^\circ\}$ and HF maps $H^r, r \in \{0^\circ, 90^\circ, 180^\circ, 270^\circ\}$ are obtained on CPU firstly, Then GPU is responsible for dividing MF maps into concatenate feature vectors $V_{MR}^i, i = 1, 2, 3, \ldots, M$ and dividing HF maps into patches $P_{HR}^i, i = 1, 2, 3, \ldots, M$.

In essence, the way to process MF maps and HF maps is same. The only difference is the number of MF maps is D times that of HF maps. We just illustrate the process of dividing an HF map into patches here. This process is parallelized, and it can be illustrated with Fig. 9. As shown in Fig. 9, each point refers to a pixel in an image, while each box refers to an image patch or a CUDA thread. Each thread is responsible for processing a patch. The first step in the kernel function is to copy data from global memory space to shared memory space for low-latency access. Since the threads with in a thread block may copy same data, it is not necessary for all threads to participate in data copying. We assign the data copying work to these special threads whose ID (ix, iy) can be divided by patch size, for example the red points shown in Fig. 9. The pseudo-code is shown in Fig. 10.

3.2.2.2 Implementation of the inferring phase In inferring phase, multi-view query MF maps are obtained on CPU firstly; Then GPU is responsible for following tasks: dividing query MF maps into concatenate feature vectors $V_{MQ}^i, i = 1, 2, 3, \ldots, N$ (step 10 in Fig. 6), searching for K most similar results in reference space for all query objects (step 11 in Fig. 6), estimating HF patches (step 12 in Fig. 6), and merging estimated HF patches into an HF map (step 13 in Fig. 6).

Query objects $V_{MQ}^i, i = 1, 2, 3, \ldots, N$ can be obtained by launching the same kernel function defined in last part. In this part, we focus on how to implement steps (11)–(13) with CUDA. The details are as follows:

Implementation of step (11) For any two given query objects V_{MQ}^i and V_{MQ}^j, searching for their K most similar patches in reference space is independent to each other, therefore this process can be fully parallelized. We exploit a CUDA-version KNN program implemented by Carcia

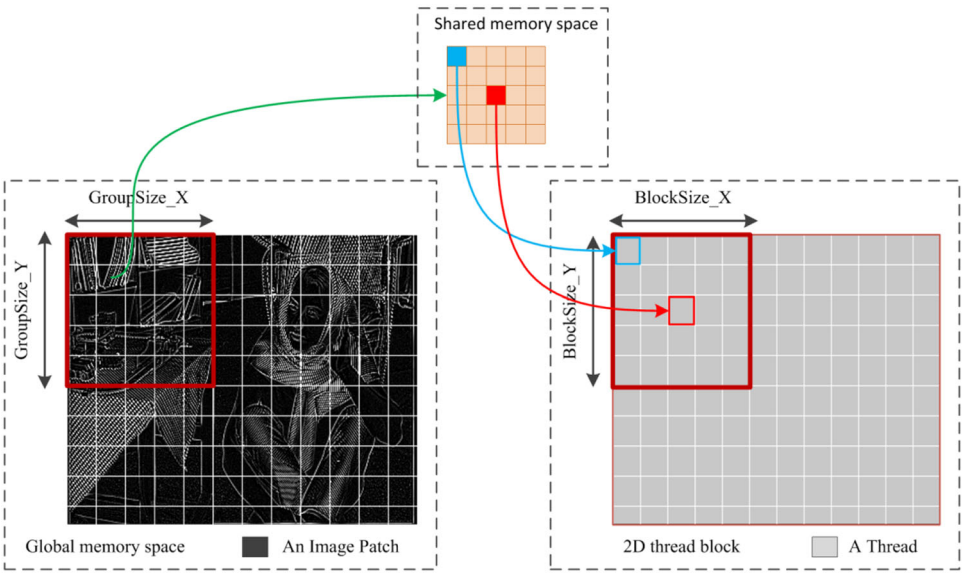


Fig. 8 Fine-grained threads parallelism and data parallelism for SR

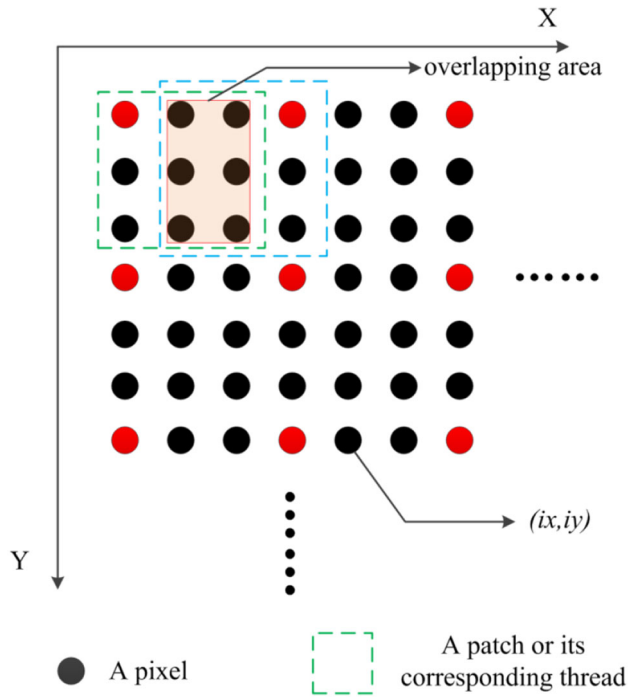


Fig. 9 The parallelism of dividing process

et al. [27] to achieve our goal of this step. There are two core steps in this process as follows:

1. Calculating the distance matrix D. Each element $D(i,j)$, which is the distance between a given query object P^j_{MQ} and a given reference object P^i_{MR}, is calculated by a CUDA thread. This process can be fully parallelized.
2. Sorting distance matrix D and index matrix I. Each column of the initialized index matrix I is a vector $(1,2,3,\ldots,M)^T$, where $(\cdot)^T$ denotes the transpose operation. Each column of distance matrix D is sorted by a CUDA thread. The index updating of distance matrix D is simultaneously applied to the index matrix I during sorting.

Implementation of step (12) With the reference HF patches $P^i_{HR}, i = 1,2,3,\ldots,M$, the index matrix I, and the distance matrix D, an estimated HF patch can be calculated by Eq. (16). Each CUDA thread within a 1D thread block is responsible for calculating corresponding estimated HF patch. Another task in this step is to record the overlapping areas for next step. The implementation details of this step can be illustrated in Figs. 11 and 12.

Implementation of step (13) The goal of this step is to calculate the value of overlapping areas in estimated HF map. Each thread within a 2D thread block is responsible for dealing with its corresponding pixel in estimated HF map. The pseudo-code of this step is shown in Fig. 13.

4 Experimental results

To validate the effectiveness of the proposed method, we conduct two groups of experiments to make a contrast with other methods in reconstruction quality and consuming time. Another group of experiments are performed to explore the effect of parameters on reconstruction results. We exploit four datasets including Set5, Set14, Urban100, and BSDS100 to perform our experiments. Some typical images in these datasets are shown in Fig. 14. Our experiments are executed on a machine with a GPU of NVIDIA GeForce GTX 980, an Intel® Core™ i7-3770 K CPU with 3.50 GHz, and 16 GB DRAM. Our program is developed

Fig. 10 Pseudo-code of dividing process

```
/* dev_Maps:input map                    */
/* dev_Patches:output patches            */
/* PatchSize: the size of image patch    */
/* BlockSize: the size of thread block   */
/* ImageWidth: width of input map        */
/* ImgaeHeight: height of input map      */
Define kernel function Divide_Kernel(dev_Map,dev_Patches,Patchsize,BlockSize,MapWidth,MapHeight)
        *allocate shared memory Temp[BlockSize][BlockSize] for this block
        *retrieve thread ID(ix,iy)
        If ix < MapWidth && iy < MapHeight
                If mod(ix,PatchSize)==0 && mod(iy,PatchSize)==0
                        For x = ix : ix+PatchSize-1
                                For y = iy : iy+PatchSize-1
                                        Temp[x][y] = dev_Map[x][y]
                                End For
                        End For
                End If
                *synchronize to wait memory copy completed
                *convert Temp[ix:ix+PatchSize][iy:iy+PatchSize] into a column vector V
                *write the column vector V to dev_Patches[1:PatchSize*PatchSize][iy*MapWidth+ix]
        End If
End define
```

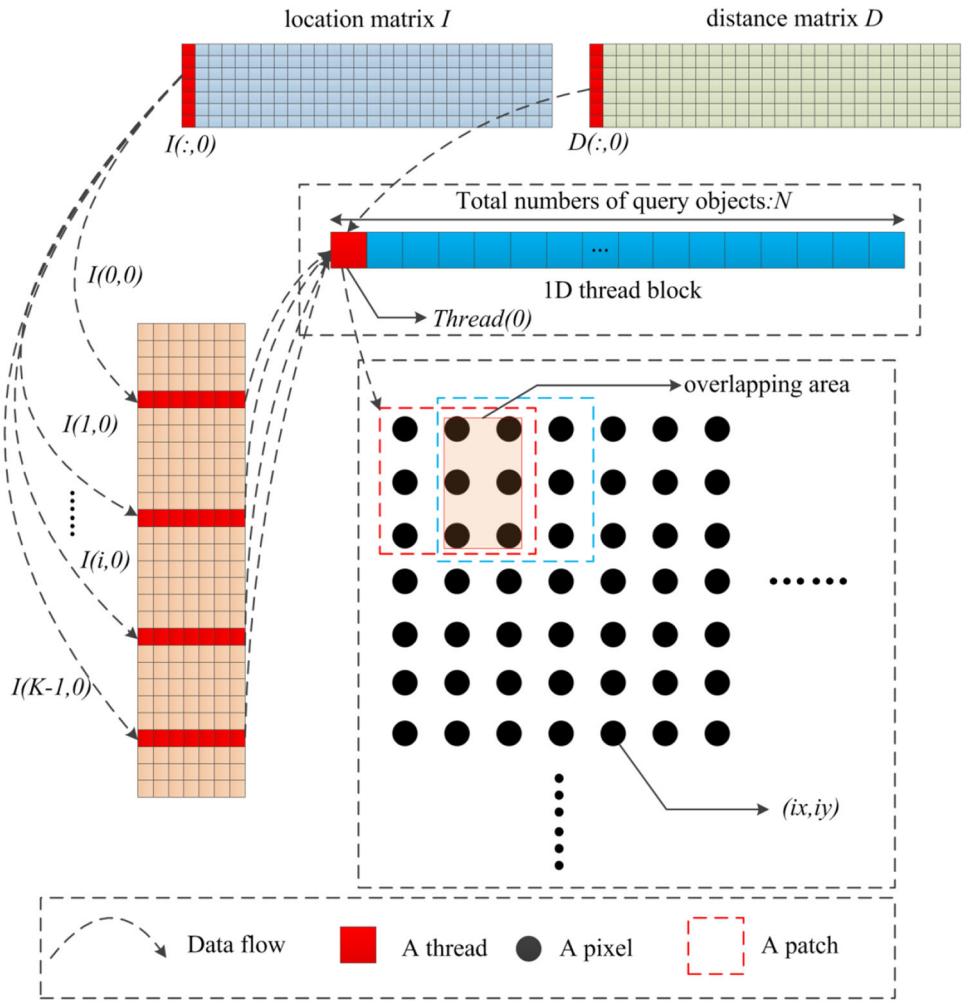


Fig. 11 Process of estimating HF map

with CUDA toolkit 7.5, and it is compiled with x64 and release options. Some key parameters in our experiments are listed in Table 1.

4.1 Reconstruction quality comparison

We apply peak signal-to-noise ratio (PSNR) and structural similarity (SSIM) as objective evaluation criteria of reconstruction quality. Table 2 lists our experimental results.

The average PSNR and SSIM of the resultant images for each dataset are shown in Table 2. Numbers in red indicate the best result, and those in blue indicate the second best result. As shown in Table 1, our method performs best on PSNR over all four datasets and performs best on SSIM over three datasets. Compared with the Wu's method, our method is improved by about 0.5 db on PSNR.

In addition to the comparison of objective evaluation criteria, we also evaluate the visual perception of reconstructed images. Figures 15, 16, 17 and 18 show four reconstructed images and their corresponding magnified local regions in red boxes. As we can see from these magnified images, images reconstructed based on bicubic interpolation have blurred edges and lost some dedicated details. Images obtained by Wu's method improved visual perception in edge areas. This would let edges of images look sharper. However, some artifacts such as ringing were introduced, especially in images with lots of tiny textures. On the contrast, images reconstructed by our method were visually closest to ground truth images.

All evaluations listed above demonstrate that our method performs best among the compared methods in both terms of visual perception and objective criteria.

4.2 Consuming time comparison

We select another three methods which use internal training set to conduct consuming time contrast experiments. Table 3 lists the average consuming time of four datasets. Numbers in red indicate the fastest result and numbers in blue indicate the second fast result. As Table 3 shows, our method performs best compared with other three methods

```
/* dev_RefHFPatches : input reference HF patches                    */
/* dev_IndexMat: intpu index matrix obtained in step(11)            */
/* dev_DistanceMat : input distance matrix obtained in step(11)     */
/* QueryWidth : the numbers of query objects                        */
/* ReferWidth : the numbers of reference objects                    */
/* PatchSize : the size of image patch                              */
/* K : the number of K most similar patches                         */
/* MapWidth : the width of estimated HF map                         */
/* dev_EstimatedHFMap : the estimated HF map(output)                */
/* dev_Overlappint: the overlapping areas(output)                   */
Define kernel function CalculateHFMap(dev_RefHFPatches,dev_IndexMat,dev_DistanceMat,QueryWidth,ReferWidth,
                                      PatchSize,K,MapWidth,dev_EstimatedHFMap,dev_Overlapping)
        *retrieve thread ID ix
        If ix < QueryWidth
                *calculate weight coefficients of K reference HF patches according to  Eq.(8)
                *normalize weight coefficients
                *calculate estimated HF patches tempHF according to Eq.(16)
                y = mod(ix,MapWidth)
                x = floor(ix/MapWidth)
                For i = x:x+PatchSize
                        For j = y:y+PatchSize
                                atomicAdd(dev_EstimatedHFMap[i][j] ,tmepHF[i-x][j-y])
                                atomicAdd(dev_Overlappingp[i][j],1)
                        End For
                End For
        End If
End define
```

Fig. 12 Pseudo-code of step (12)

```
/* dev_EstimatedHFMap : the estimated HF map */
/* dev_Overlapping : the overlapping areas   */
/* MapWidth : the width of estimated HF map  */
/* MapHeight : the height of estimated HF map */
Define kernel function CalculateOverlapping(dev_EstimatedHFMap,dev_Overlapping,MapWidth,MapHeight)
        *retrieve thread ID (ix,iy)
        If ix < ImageWidth && iy < ImageHeight
                If dev_overlappint[ix][iy] == 0
                        dev_Overlappint[ix][iy] = 1
                End If
                dev_HFMap[ix][iy] = dev_HFMap[ix][iy] / dev_Overlapping[ix][iy]
        End If
End define
```

Fig. 13 Pseudo-code of step (13)

Fig. 14 Typical images in datasets

over all four datasets. Speedup ratio compared with other three methods is shown in Fig. 19. A 12x speedup can be gained compared with second fastest method on Set5, and the 2x speedup on Urban100 is the minimum speedup ratio.

Table 1 Key parameters in our experiments

Upscale factor	2
Patch size	3×3
Overlap length among patches	2
The number of K most similar patches	10
The number of DoG filters	3

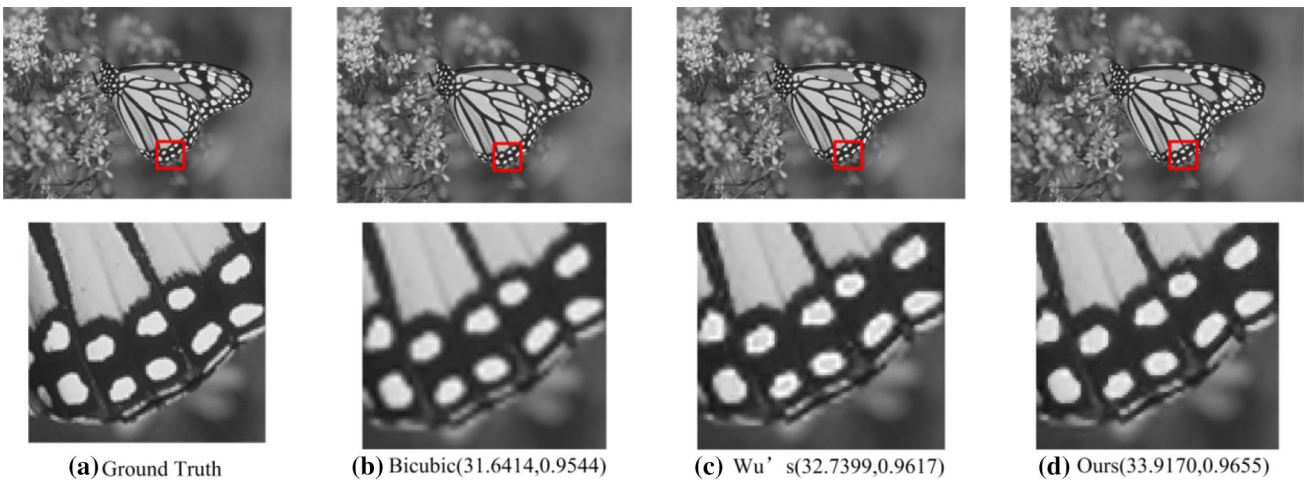
The reason why long consuming time on Urban100 is that the average size of images in Urban100 is larger than that of other datasets.

4.3 The effect of parameters

We select six images to conduct experiments in this part, as shown in Fig. 20. In order to perform experiments

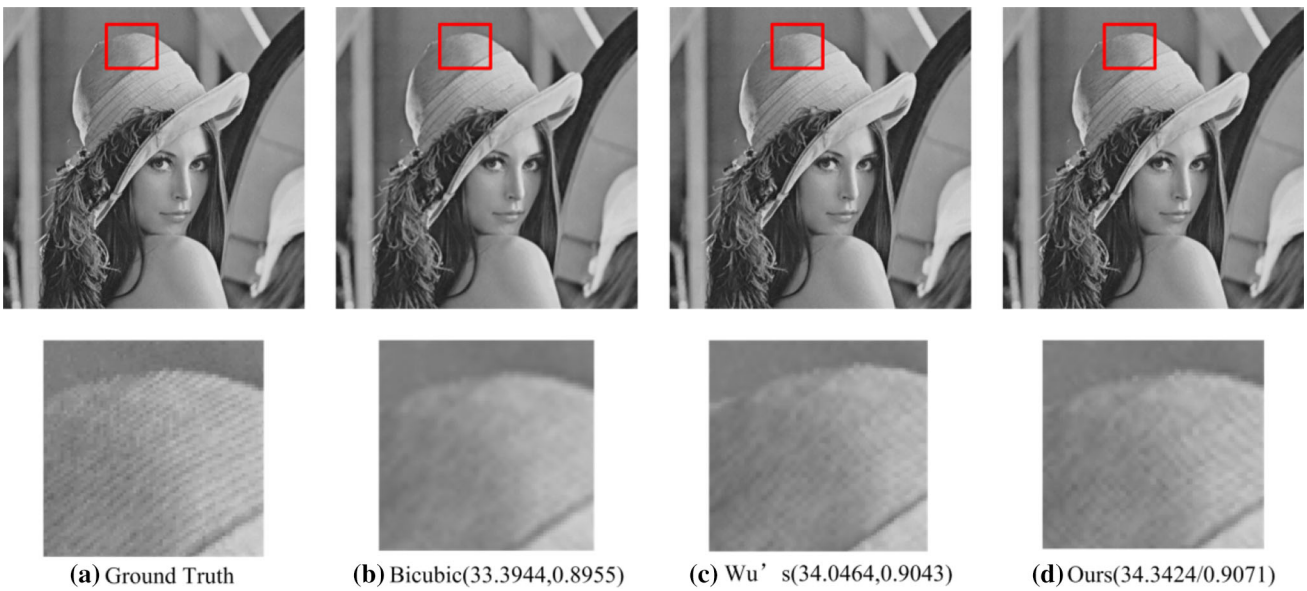
Table 2 Reconstruction quality (PSNR, SSIM) comparison

Method\dataset	Set5	Set14	Urban100	BSDS100
Bicubic	32.3423, 0.9219	28.7749, 0.8580	25.5425, 0.8306	28.2434, 0.8322
Wu's [22]	33.1182, 0.9335	28.8086, 0.8691	25.8918, 0.8499	28.6039, 0.8513
Our method	33.6618, 0.9380	29.2129, 0.8688	26.7657, 0.8649	28.7887, 0.8556



(a) Ground Truth **(b)** Bicubic(31.6414,0.9544) **(c)** Wu' s(32.7399,0.9617) **(d)** Ours(33.9170,0.9655)

Fig. 15 Experimental results on Monarch in Set14



(a) Ground Truth **(b)** Bicubic(33.3944,0.8955) **(c)** Wu' s(34.0464,0.9043) **(d)** Ours(34.3424/0.9071)

Fig. 16 Experimental results on Lenna in Set14

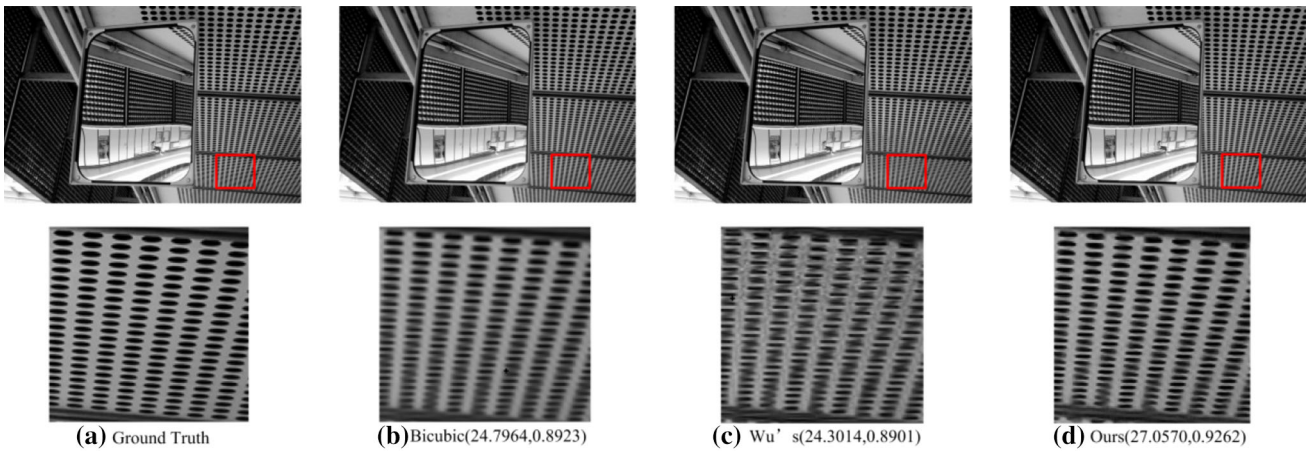


Fig. 17 Experimental results on 004 in Urban100

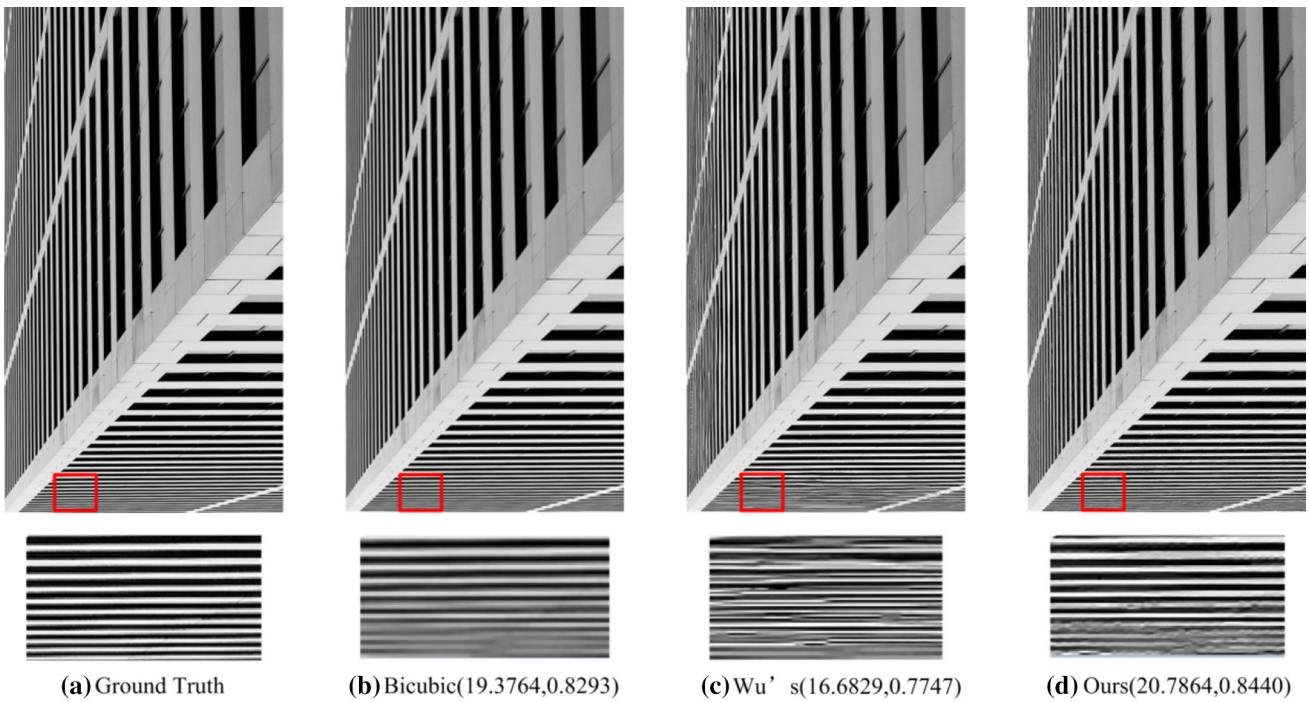


Fig. 18 Experimental results on 011 in Urban100

Table 3 Consuming time comparison (unit: s)

Method\dataset	Set5	Set14	Urban100	BSDS100
Wu's [22]	128.798	254.915	829.729	159.556
Huang's [13]	26.836	62.198	245.836	37.577
Zeyde's [8]	21.159	40.640	140.453	27.745
Ours	1.710	4.890	69.100	2.412

conveniently, we choose the program without acceleration to conduct experiments in this part.

4.3.1 The effect of expanding training set

We conduct two experiments is this part. The one rotates input LR image to build internal training set, while the other one does not. Table 4 lists the experimental results on the test images and the better results are marked with bold numbers. Table 4 indicates that rotating input LR image can improve reconstruction quality.

4.3.2 The effect of the number of DoG filter

We conduct six experiments and each experiment use DoG filter with different numbers. Figure 21 shows the average

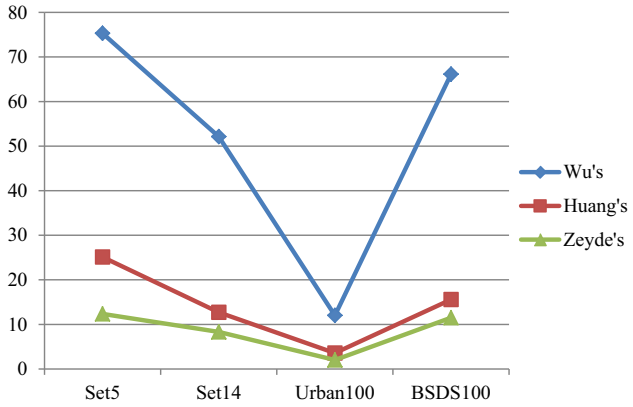


Fig. 19 Speedup ratio

Table 4 Reconstruction quality as function of whether expanding training set

Image\method	Without rotating LR image PSNR (dB), SSIM	Rotating LR images PSNR (dB), SSIM
Barbara	27.2513,0.8574	**27.3141**, **0.8587**
Bird	37.3815,0.9784	**38.2643**, **0.9811**
Foreman	32.8328,0.9584	**33.0580**, **0.9592**
Lenna	34.5871,0.9081	**34.8509**, **0.9109**
Pepper	32.6836,0.8995	**33.7804**, **0.9017**
PPT	**27.8005**, **0.9679**	27.7736, 0.9652
Average	32.08947,0.928283	**32.50688**, **0.929467**

PSNR and SSIM on test images of each experiment. Experimental results demonstrate the proposed method performs better as the number of DoG filter increases. When the number is bigger than 4, there is no significant improvement in reconstruction quality.

4.3.3 The effect of patch size

We conduct five experiments to explore the effect of patch size. Figure 22 shows reconstruction quality as function of patch size. As shown in Fig. 22, the reconstruction quality curve first rises and then drops. The reason for these phenomena is that the smaller the patch size is, the less information each patch could contain. When the patch size equals to 1×1, the proposed method is almost degraded to an interpolation-based method. However, if the patch size is too large, artifacts in reconstructed images would be exacerbated.

Fig. 20 Images used in exploring effect of parameters

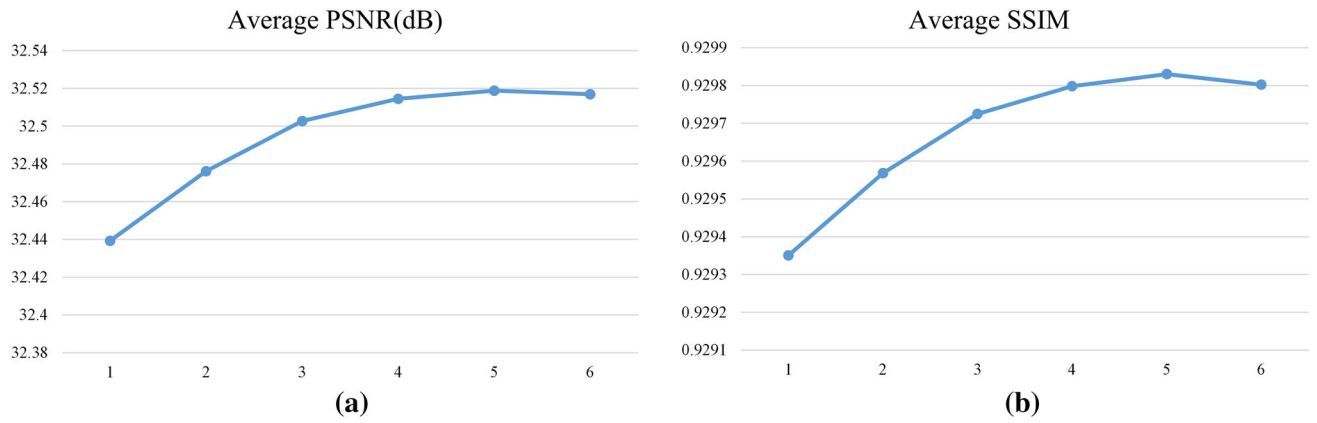


Fig. 21 Reconstruction quality as function of DoG filter numbers

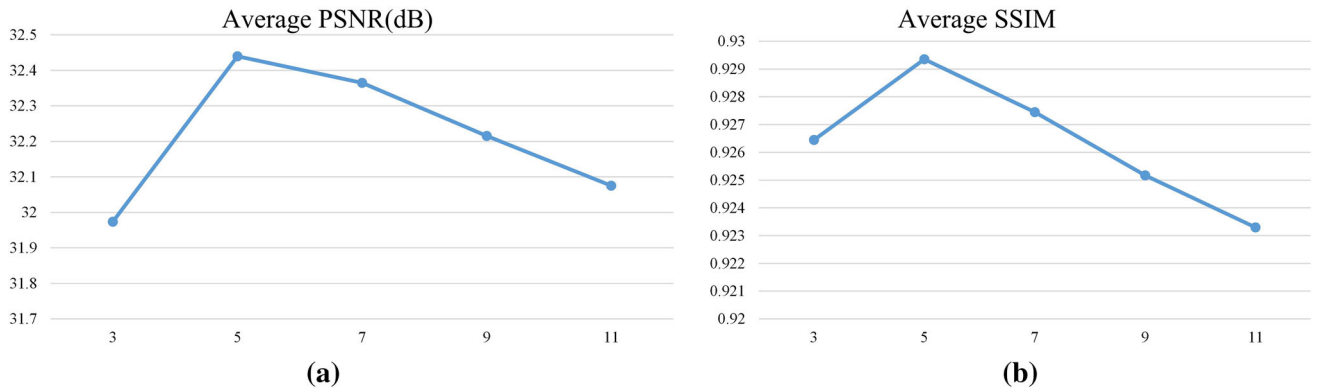


Fig. 22 Reconstruction quality as function of patch size

4.3.4 The effect of the number of *K* most similar patches

We conduct a set of experiments to explore the effect of the number of *K* most similar patches. The trend of PSNR and SSIM is shown in Fig. 23. As shown in Fig. 23, the trend curve first rises and then drops. The reason for these phenomena is that the more numbers of similar patches participating in estimating HF patches, the more useful prior information would be integrated in. However, excessive similar patches could not give useful prior information as their correlation becoming weak.

5 Conclusion

In this paper, we proposed a learning-based SR method which does not need external training set. The proposed method has two phases: the training phase and the inferring phase. In the training phase, the input LR image is rotated with different angles to enhance the prior information. Multi-scale DoG filters are exploited to multi-view feature maps, which provide accurate representation of images. An internal training set is built by taking advantage of self-similarity of images. In the inferring phase, NLM is exploited to calculate the estimated HF details. Experimental results demonstrate that the proposed method performs best in terms of reconstruction quality among the compared methods. In order to meet the requirement of real time and/or near real time, we implemented the proposed method with CUDA for high-execution effectiveness. Experimental results demonstrate that the proposed method runs faster than other internal training set-based methods. The proposed method may be readily suitable for real-time and/or near-real-time processing applications.

For our further work, advanced methods such as image transform can be exploited to expand training set. Moreover, multi-type filters can be exploited to represent images more accurately. In terms of execution effectiveness, extracting feature map from image can be parallelized to further improve running speed.

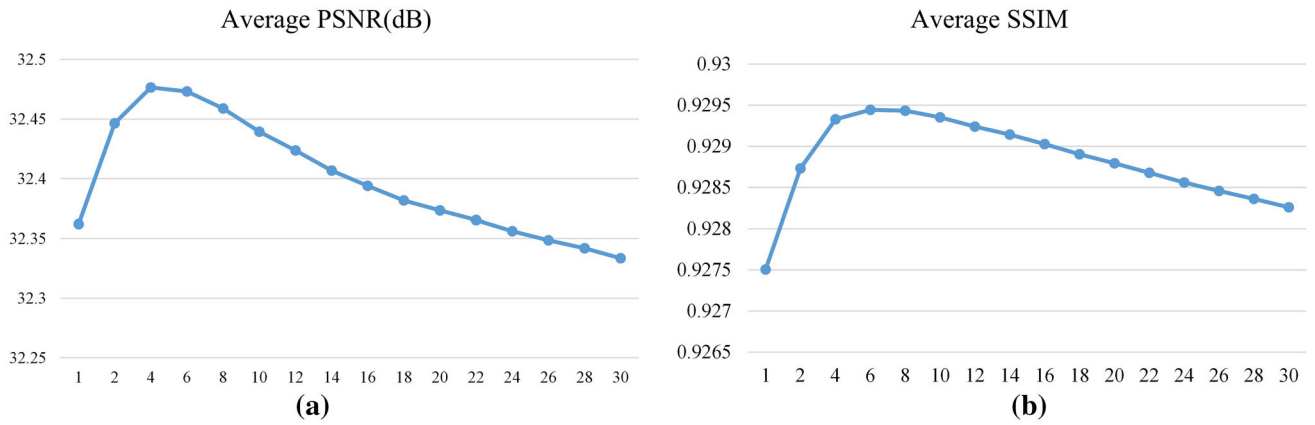


Fig. 23 Reconstruction quality as function of the number of K most similar patches

Acknowledgements The research in our paper is sponsored by National Natural Science Foundation of China (Nos. 61701327, 61711540303, 61473198), also is supported by the Priority Academic Program Development of Jiangsu Higher Education Institutions (PAPD) Fund, Jiangsu Collaborative Innovation Center on Atmospheric Environment and Equipment Technology (CICAEET) Fund.

References

1. Park, S.C., Park, M.K., Kang, M.G.: Super-resolution image reconstruction: a technical overview. IEEE Signal Process. Mag. **20**(3), 21–36 (2003)
2. Li, X., Hu, Y., Gao, X., Tao, D., Ning, B.: A multi-frame image super-resolution method. Sig. Process. **90**(2), 405–414 (2010)
3. Aguena, M.L., Mascarenhas, N.D.: Multispectral image data fusion using POCS and super-resolution. Comput. Vis. Image Underst. **102**(2), 178–187 (2006)
4. Qin, F.Q., He, X.H., Chen, W.L., Yang, X.M., Wu, W.: Video superresolution reconstruction based on subpixel registration and iterative back projection. J. Electron. Imaging **18**(1), 013007 (2009)
5. Vrigkas, M., Nikou, C., Kondi, L.P.: Accurate image registration for MAP image super-resolution. Sig. Process. Image Commun. **28**(5), 494–508 (2013)
6. Glasner, D., Bagon, S., Irani, M.: Super-resolution from a single image. In: 2009 IEEE 12th International Conference on Computer Vision, pp. 349–356. IEEE (2009)
7. Kim, K.I., Kwon, Y.: Single-image super-resolution using sparse regression and natural image prior. IEEE Trans. Pattern Anal. Mach. Intell. **32**(6), 1127–1133 (2010)
8. Zeyde, R., Elad, M., Protter, M.: On single image scale-up using sparse-representations. In: *International Conference on Curves and Surfaces*, pp. 711–730. Springer, Berlin (2010)
9. Yang, J., Wright, J., Huang, T.S., Ma, Y.: Image super-resolution via sparse representation. IEEE Trans. Image Process. **19**(11), 2861–2873 (2010)
10. Yang, X., Wu, W., Liu, K., Chen, W., Zhang, P., Zhou, Z.: Multi-sensor image super-resolution with fuzzy cluster by using multi-scale and multi-view sparse coding for infrared image. Multimed. Tools Appl. **76**(2), 1–32 (2017)
11. Wu, W., Yang, X., Liu, K., Liu, Y., Yan, B.: A new framework for remote sensing image super-resolution: sparse representation-based method by processing dictionaries with multi-type features. J. Syst. Architect. **64**, 63–75 (2016)
12. Dong, C., Loy, C.C., He, K., Tang, X.: Learning a deep convolutional network for image super-resolution. In: European Conference on Computer Vision, pp. 184–199. Springer, Cham (2014, September)
13. Huang, J.B., Singh, A., Ahuja, N.: Single image super-resolution from transformed self-exemplars. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5197–5206 (2015)
14. Zhou, Z., Wang, Y., Wu, Q.J., Yang, C.N., Sun, X.: Effective and efficient global context verification for image copy detection. IEEE Trans. Inf. Forensics Secur. **12**(1), 48–63 (2017)
15. Li, J., Li, X., Yang, B., Sun, X.: Segmentation-based image copy-move forgery detection scheme. IEEE Trans. Inf. Forensics Secur. **10**(3), 507–518 (2015)
16. Pan, Z., Zhang, Y., Kwong, S.: Efficient motion and disparity estimation optimization for low complexity multiview video coding. IEEE Trans. Broadcast. **61**(2), 166–176 (2015)
17. Rong, H., Ma, T., Tang, M.: A novel subgraph K+-isomorphism method is social network based on graph similarity detection. Soft. Comput. (2017). https://doi.org/10.1007/s00500-017-2513-y
18. Zhou, Z., Wu, Q.J., Huang, F., Sun, X.: Fast and accurate near-duplicate image elimination for visual sensor networks. Int. J. Distrib. Sens. Netw. **13**(2), 1550147717694172 (2017)
19. Zhou, Z., Yang, C.N., Chen, B., Sun, X., Liu, Q., Qm, J.: Effective and efficient image copy detection with resistance to arbitrary rotation. IEICE Trans. Inf. Syst. **99**(6), 1531–1540 (2016)
20. Yang, C.Y., Huang, J.B., Yang, M.H.: Exploiting self-similarities for single frame super-resolution. In: Asian Conference on Computer Vision, pp. 497–510. Springer, Berlin (2010)
21. Suetake, N., Sakano, M., Uchino, E.: Image super-resolution based on local self-similarity. Opt. Rev. **15**(1), 26–30 (2008)
22. Wu, W., Zheng, C.: Single image super-resolution using self-similarity and generalized nonlocal mean. In: TENCON 2013-2013 IEEE Region 10 Conference, pp 1–4. IEEE (2013)
23. Sun, K.Z., Li, J.D., Xu, S.Y.: Gpu-Accelerated non-local means super-resolution reconstruction. In: Proceedings of the 3rd International Conference on Multimedia Technology, ICMT, pp. 1242–1249 (2013)
24. Wang, Y.K., Huang, W.B.: A CUDA-enabled parallel algorithm for accelerating retinex. J. Real Time Image Proc. **9**(3), 407–425 (2014)
25. Document Page of Nvidia. http://docs.nvidia.com/cuda/cuda-c-best-practices-guide. Accessed 18 Dec 2017
26. Home Page of Nvidia. https://developer.nvidia.com/. Accessed 18 Dec 2017

27. Garcia, V., Debreuve, E., Nielsen, F., Barlaud, M.: K-nearest neighbor search: fast GPU-based implementations and application to high-dimensional feature matching. In: 2010 17th IEEE International Conference on Image Processing (ICIP), pp. 3757–3760. IEEE (2010)

Yuan Yuan is currently pursuing master degree in College of Electronics and Information Engineering, Sichuan University. His research interests are image process and computer vision.

Xiaomin Yang is currently an Associate Professor in College of Electronics and Information Engineering, Sichuan University. She received her BS degree from Sichuan University, and received her PhD degree in communication and information system from Sichuan University. She worked in University of Adelaide as a postdoctorate for one year. Her research interests are image processing and pattern recognition.

Wei Wu is currently an Associate Professor in College of Electronics and Information Engineering, Sichuan University. He received his BS degree from Tianjin University, and received his MS and PhD degrees in communication and information system from Sichuan University. He worked in a National Research Council, Canada, as a postdoctorate for one year. His research interests are image processing and pattern recognition.

Hu Li is currently pursuing master degree in College of Electronics and Information Engineering, Sichuan University. His research interests are image process and computer vision.

Yiguang Liu received the MS degree from Peking University, in 1998, and the PhD degree from Sichuan University, in 2004. He was a Research Fellow, Visiting Professor, and Senior Research Scholar with the National University of Singapore, Imperial College London, and Michigan State University, respectively. He was chosen into the program for new century excellent talents of MOE in 2008, and chosen as a Scientific and Technical Leader in Sichuan Province in 2010.

Kai Liu is currently a professor in the School of Electrical Engineering and Information at Sichuan University, China. He received his BS and MS degrees in computer science from Sichuan University, and his PhD in electrical engineering from the University of Kentucky, respectively. His main research interests include computer/machine vision, active/passive stereo vision and image processing. He is a senior member of the IEEE.