Memory optimized Deep Dense Network for Image Super-resolution

Jialiang Shen*[2], Yucheng Wang*[1], and Jian Zhang[2]

[1]Intelligent Driving Group, Baidu Inc
[2]Multimedia and Data Analytics Lab, University of Technology Sydney
jialiang.shen@student.uts.edu.au, wangyucheng@baidu.com, jian.zhang@uts.edu.au

***Abstract*—CNN methods for image super-resolution consume a large number of training-time memory, due to the feature size will not decrease as the network goes deeper. To reduce the memory consumption during training, we propose a memory optimized deep dense network for image super-resolution. We first reduce redundant features learning, by rationally designing the skip connection and dense connection in the network. Then we adopt share memory allocations to store concatenated features and Batch Normalization intermediate feature maps. The memory optimized network consumes less memory than normal dense network. We also evaluate our proposed architecture on highly competitive super-resolution benchmark datasets. Our deep dense network outperforms some existing methods, and requires relatively less computation.**

***Index Terms*—Image super-resolution, Dense connection, Memory-optimized**

I. Introduction

The process of reconstructing high-resolution (HR) images from their low-resolution images (LR) is referred to as super-resolution. SR has a wide range of computer vision applications, such as remote sensing satellite imaging [17], medical image processing [4], microscope image processing [22], multimedia industry [21], and surveillance [15] where many SR based works are involved.

Single Image SR problem is an underdetermined inverse problem, and the solution for the problem is not unique. Early methods include prediction methods [3] generating HR pixels intensities by weighted averaging neighbouring LR pixel values, edge based methods which learn priors from edge features [23] for reconstructing HR images, image statistical methods predicting HR images from LR images using various image properties [18]. Currently, deep learning methods are widely used to learn a mapping from LR to HR patches.

The pioneer CNN model for SR [1] learns the mapping between input LR image and corresponding HR output via three convolutional layers and archives superior performance to classical non-deep learning methods. However, SRCNN [1] fails to achieve better performance when trains with deeper structures, which is due to the gradient vanishing problem in deeper networks. Residual learning is proposed in many publications [5] [6] [7] to address this problem by adopting skip connection between layers. However, some studies [24] have found residual network behaves like ensembles of relatively shallow networks and only shallow paths in residual network contribute to the gradient during training. Furthermore residual blocks usually consists of only two convolutional layers, which restricts learning deeper and more expressive features.

Recently, a new dense architecture [19] was introduced in image super-resolution and achieved great success in terms of both reconstruction accuracy and computational performance. It is due to each layer is connected to all the other layers in the dense blocks, rather than only connected to one early layers in residual blocks. These connections promote feature reuse that early-layer features can be utilized by all other layers. The characteristics of dense architecture make it a very good fit for image super-resolution as they naturally induce skip connections. Tong [14] proposed SRDenseNet, using dense network for image super-resolution by removing pooling players and transition layers. Zhang et al. [13] also introduces dense blocks in RDN. Compared to SRDenseNet [14], RDN [13] uses larger growth rates to construct a wider network for further improving the performance. But we find directly applying dense block in image super-resolution will generate a large number of network parameters, this is because each layer in dense block uses all previous feature maps as input. As a result, the number of parameters increases quadratically with network depth. And the construction of deep network is commonly limited by GPU memory.

To reduce the training-time memory, many researchers [25] [26] [27] find that when training a deep convolutional network, a large proportion of memory is used to store the intermediate outputs and backward gradients. Chen [25] implement a 1000 layer memory-efficient ResNet [8] model using one GPU by dropping the intermediate results into share memory allocations during training. The ResNet model is construced by composite Conv-BN-Relu [8] layers, so the algorithm only keep the result of convolution, but drop the result of batch normalization and activation function within each composite layer. The dropped results are recomputed before each composite layer back-propagation. Plesis et al [27], observe the feature reuse mechanism of DenseNet [19] causes large memory consumption. The intermediate feature maps in DenseNet [19], such as Batch Normalization and Concatenation, are responsible for most of the memory con-

* equal contribution

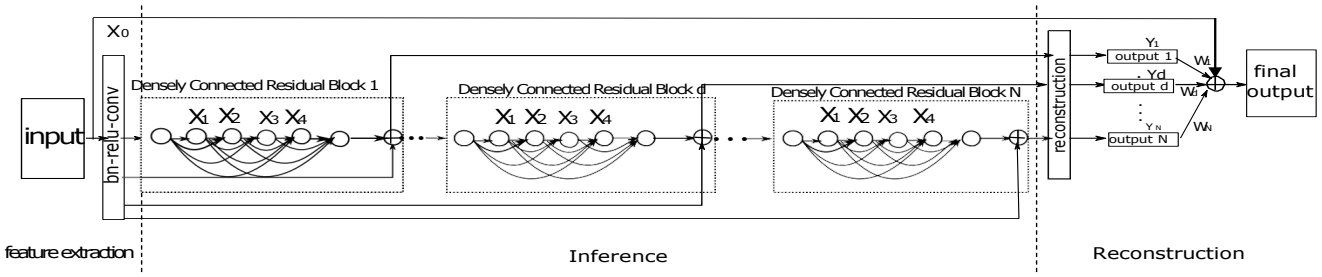

Fig. 1. Our network structure with densely connected residual block and deep supervision.

sumption and can be recomputed in share memory allocations before back-propagation. However, these models are only designed for image classification, which is not suitable for image super-resolution tasks. Furthermore, deep learning methods for image super-resolution require more GPU memory than image classification since the size of feature maps will not decrease as they are propagated to the network end. Therefore it is very important to have memory optimized training algorithms for image SR. In order to efficiently use GPU memory to train network, we implement a memory optimized deep dense network for image super-resolution. On the one hand, we rationally design layer connections in the blocks to reduce redundant features learning and minimize the model size. On the other hand, we adopt share memory allocation strategy to store the concatenated features and Batch Normalization intermediate feature maps to reduce the training memory cost of network. The proposed network reaches a promising performance on the benchmark datasets with less training-time memory.

II. Methods

In this section, we first describe the network structure, then introduce our densely connected residual block, and share memory allocation for memory optimized deep dense network.

A. Network Structure

Our network configuration is outlined in Fig. 1. The model consists of three sub-network structures: feature extraction, Inference, and reconstruction. Feature extraction sub-architecture is used to extract feature maps from the low-resolution images and comprise a set of feature maps into a high dimensional vector. And then inference is aimed to expand the high dimensional vector into multiple channels vectors and construct a deep network structure for the task. Finally, the reconstruction sub-architecture is to generate the output image.

1) Feature Extraction Sub-architecture: It extracts patches from the interpolated input image X. The interpolated images are upscaled by bicubic interpolation as the low-resolution images. In our feature extraction sub-architecture, we implement a composite function $H_0()$ of three consecutive operations: batch normalization (BN) [11], rectified liner unit (ReLU) [12] and convolutional filters (Conv) to extract features and represent them in the form of vectors. Considering a single low-resolution image (interpolated image) as input X, the feature extraction sub-architecture output as X_0, and X_0 is formulated as below:

$$X_0 = H_0(X) \tag{1}$$

2) Inference Sub-architecture: The inference sub-architecture is the main part to learn the mapping from the LR features to HR features. In the inference sub-architecture, we use our densely connected residual blocks to learn features and stack them into a chained network. Details about densely connected residual blocks will be given in the following subsection. Fig. 1 illustrates the layout of the dense connectivity in our inference sub-architecture. Denote the inference sub-architecture contains N densely connected residual blocks, and the function of i_{th} densely connected residual block as $D_i()$. Due to inference sub-architecture is equivalent to the composition of the chained densely connected residual blocks, we formulate the output of inference sub-architecture Y^N as:

$$Y^N = D_N(D_{N-1}(...(D_1(X_0))...)) \tag{2}$$

3) Reconstruction Sub-architecture: The reconstruction layer is a 1-channel convolutional layer. Each densely connected residual block will generate their own reconstructed images. We then sum the averaged reconstructed images as the output of reconstruction sub-architecture. Denote the output of reconstructing the i_{th} densely connected residual block as Y_i

$$Y_i = f_{res}(D_i(D_{i-1}(...(D_1(X_0))...))) \tag{5}$$

Here, $D_i()$ and $f_{res}()$ are the functions of the i_{th} densely connected residual block and reconstruction. Denoting $\widehat{Y}$ as the reconstruction image, which is produced by adding up the averaged reconstructed images with the input X. We formulate the reconstruction image as below:

$$\widehat{Y} = \frac{Y_1 + .. + Y_i + .. + Y_N}{N} + X \tag{6}$$

Here we have N densely connected residual blocks to construct the network.

B. Densely Connected Residual Block

Fig. 2 illustrates the layout of the dense connectivity in densely connected residual block. The ℓ_{th} layer in the block receives the feature channels of all preceding layers $X_0, ..., X_{\ell-1}$

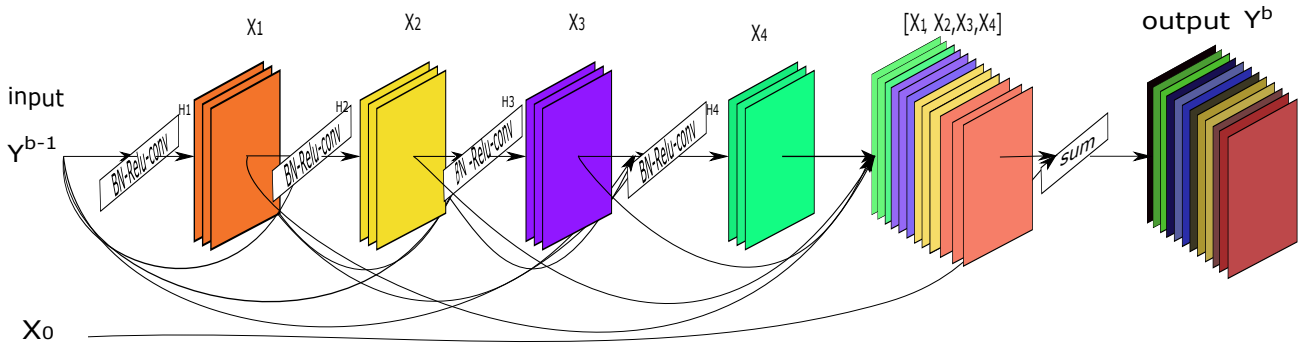

Fig. 2. Densely connected residual block

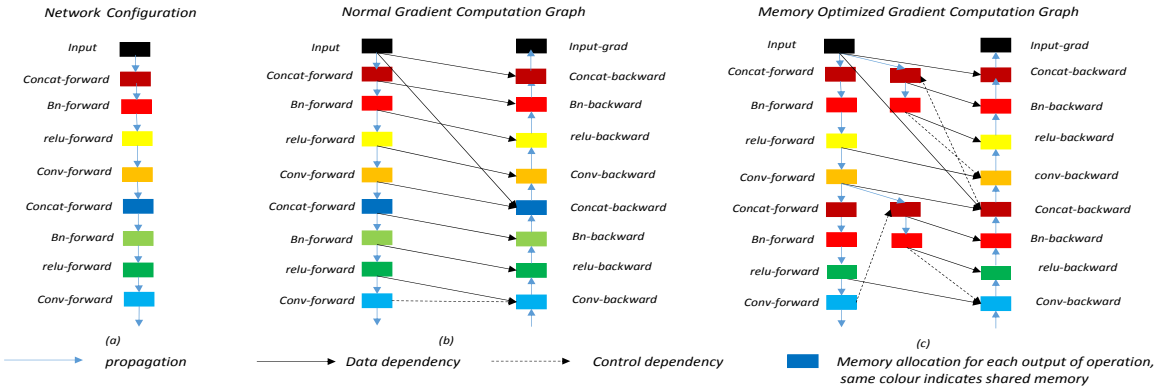

Fig. 3. Computation graph of memory optimized network

as input.

$$X_\ell = H_\ell([X_0, X_1, ..., X_{\ell-1}]) \tag{3}$$

where [] refers to the concatenation function, and H_ℓ is the ℓ_{th} layer composite function.

To reduce computation redundancy and enhance feature reuse, we first keep the extracted feature X_0 and concatenated channel layers $[X_1, ..., X_4]$ the same channel size. Then sum them up to get the input for next densely connected residual block. This skip connection between X_0 and the output of the block takes advantage of residual learning. Denote Y^i as the output of i_{th} densely connected residual block. We formulate i_{th} densely connected residual block output as below:

$$Y^i = [X_1^i, ..., X_4^i] + X_0 \tag{4}$$

Our densely residual block has two advantages: (1) it delves the network potential through feature reuse and helps the gradient backpropagate due to the direct connection structure. (2) replacing transition layers with summation largely reduces the network parameters and keeps the input feature information throughout the layers.

C. Share Memory allocation

In the dense network, the intermediate features are mostly generated from concatenation and batch normalization. So we create share memory allocations for the output of concatenation and batch normalization.

The network is constructed by composite layers Concat-BN-ReLU-Conv as shown in Fig. 3(a). The normal plain dense network keep these intermediate features allocated in different GPU memory for use during back-propagation. As we can see in Fig. 3(b), the back propagation is triggered by the last convolution layer forward propagation. Then the back propagation is processed in the reverse order of forward propagation to calculate the layer parameters, input, and output gradients and update the layer parameters depending on each layer input.

We use share memory allocations for each concatenation and batch normalization. Instead of allocating new memory for concatenating the existing features, we copy pre-processed features into one share memory storage to concatenate these features into one tensor as the input for next layer. For batch

TABLE I
Public benchmark test results (PSNR(dB)/SSIM). Red indicates the best performance and blue indicates the second best.

Datasets	Scale	Bicubic	SRCNN	VDSR	DRCN	DRRN	MODN(ours)
Set5	2×	33.66/0.9929	36.66/0.9542	37.53/0.9587	37.63/0.9588	**37.74/0.9591**	**37.74/0.9593**
	3×	30.39/0.8682	32.75/0.9090	33.66/0.9213	33.82/0.9226	**34.03/0.9244**	**34.07/0.9248**
	4×	28.42/0.8104	30.48/0.8628	31.35/0.8838	31.53/0.8854	**31.68/0.8888**	**31.72/0.8889**
Set14	2×	30.24/0.8688	32.42/0.9063	33.03/0.9124	33.04/0.9118	**33.23/0.9136**	**33.25/0.9141**
	3×	27.55/0.7742	29.28/0.8209	29.77/0.8314	29.76/0.8311	**29.96/0.8349**	**29.97/0.8349**
	4×	26.00/0.7027	27.49/0.7503	28.01/0.7674	28.02/0.7670	**28.18/0.7720**	**28.25/0.7721**
BSD100	2×	29.56/0.8431	31.36/0.8879	31.90/0.8960	31.85/0.8942	**32.05/0.8973**	**32.06/0.8978**
	3×	27.21/0.7385	28.41/0.7863	28.82/0.7976	28.80/0.7963	**28.95/0.8004**	**28.96/0.8011**
	4×	25.96/0.6675	26.90/0.7101	27.29/0.7251	27.23/0.7233	**27.38/0.7284**	**27.42/0.7286**
Urban100	2×	26.88/0.8403	29.50/0.8946	30.76/0.9140	30.75/0.9133	**31.23/0.9188**	**31.27/0.9191**
	3×	24.46/0.7349	26.24/0.7989	27.14/0.8276	27.15/0.8276	**27.53/0.8378**	**27.55/0.8384**
	4×	23.14/0.6577	24.52/0.7221	25.18/0.7510	25.14/0.7510	**25.44/0.7638**	**25.52/0.7649**

normalization, we first calculate the mean and variation value of the input features, and then put the calculated batch normalization result into the share memory allocation. As shown in Fig. 3(c), the share memory allocations for concatenation and BN are used by all dense layers, so data in the storages is changing. Therefore, after the last convolutional layer forward, the concatenation and BN features are recomputed to restore the last composite layer corresponding intermediate features. After the storages storing the right data, back propagation for the last composite layer is used in the regular way. Then after the last composite layer back-propagation is done, the former layer intermediate features will be recomputed to trigger the back-propagation of the composite layer. We also analyze the memory efficient property of using share memory allocation during training in the experiment section, share memory allocation largely reduces the memory demands for training the deep network in image SR.

III. Experiment

In this section, we evaluate the performance of our model on several datasets. A description of the datasets is first provided, followed by the introduction of the implementation details. Then we analyze the depth and memory consumption of the model. After that, comparisons with state-of-art results are presented.

A. Datasets

The training dataset contains 291 images, where 91 images are from Yang et al [18], and other 200 images are from Berkeley Segmentation Dataset [29]. For testing, we use four datasets 'Set5' [20], 'Set14' [18], 'BSD100' [30] and 'Urban100' [31], which contains 5, 14 ,100 and 100 images respectively.

B. Implementation Details

Before the model setting, we first augment the training dataset the 291 images into 2328 images with flipping horizontally and rotating the training images by 90^o, 180^o, 270^o, which provide 7 additional augmented images for each original image. In addition to that, the training data were processed with multiple scale ($\times 2$, $\times 3$ and $\times 4$) augmentation by following [6] [7] [5].

We use 25 densely connected residual blocks to construct the training network. Each block contains 4 convolutional layers with 34 filters of the size 3×3. Training images are split into 31×31 patches with stride 21, so the receptive field is 31×31. We set the momentum parameter to 0.9 and weight decay to 0.0001 and 64 patches are used as a mini-batch for stochastic gradient descent.

The learning rate is initially set to 0.1 and then decreased half every 10 epochs. Since the initial learning rate is large in our design, we use the adjustable gradient clipping [6] to avoid exploding gradients. We clip the gradients to $[-\frac{\theta}{\gamma}, \frac{\theta}{\gamma}]$, where γ denotes the current learning rate and gradient clipping parameter $\theta = 0.01$. The adjustable gradient clipping makes the convergence procedure fast, our 25 blocks network takes about 4 days to train with one Titan X GPU.

C. Comparisons with State-of-Art methods

In this section, we provide quantitative and qualitative comparisons. Since the training datasets also influence performance, We use SRCNN [1], VDSR [6], DRCN [7] and DRRN [5] as the compared methods, which use the same training data as ours. In deploying trained models with test datasets, the luminance components of the image are applied for model deploying and the color components are applied with bicubic interpolation, which is the same as the previous experiments do. We also crop the image to the same size as [1] [6] [7] [5] methods.

Table. I illustrates a summary of quantitative evaluation on several datasets. Generally, our model outperforms all the other methods in both PSNR and Structural SIMilarity (SSIM) [10]. In terms of PSNR, our model achieves an average 0.2 dB improvement over VDSR and DRCN. In comparison with DRRN, our model demonstrates an average 0.03 dB increase of PSNR than DRRN. Fig. 2 shows qualitative comparisons among SRCNN, VDSR, DRRN and ours, we can see our network outperforms all the methods in the experimental

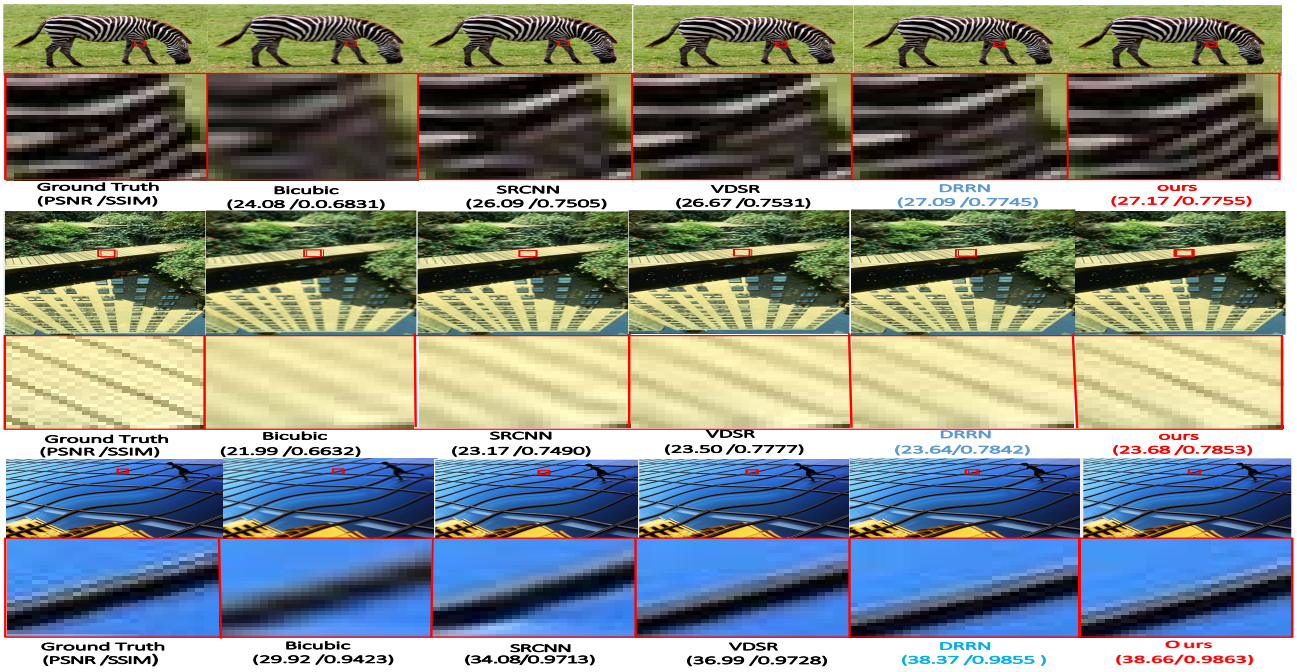

Fig. 4. Qualitative Comparison.(1)In the first row image "img014"(Set14 with scale factor $\times 4$), the zebra texture is more detailed in our method.(2)In the second row shows image"img021"(B100 with scale factor $\times 3$), the black line is clearer in our method (3) In the third row image "img082" (Urban100 with scale factor $\times 2$), we can see that is two line in ours.

PSNR and SSIM index in the most cases. For qualitative comparisons among SRCNN, VDSR, DRRN and ours, our network obviously produces patterns with sharper and cleaner edges compared with SRCNN and VDSR. In comparison with DRRN, our performance is slightly better. However, The computation complexity using our method is 20 percent less than DRRN. For 25 blocks, Our network has 11 billion FLOPs, while DRRN contains 14 billion FLOPs. This is because only 34 feature channels used in each convolutional layer in our network, while the convolutional layer in DRRN [5] has 128 feature channels.

TABLE II
The depth analysis of network

Datasets	Scale	MODN-9B	MODN-25B
Set5	2×	37.67/0.9590	37.74/0.9593
	3×	33.95/0.9234	34.07/0.9248
	4×	31.56/0.8867	31.72/0.8889
Set14	2×	33.21/0.9134	33.25/0.9141
	3×	29.91/0.8329	29.97/0.8349
	4×	28.18/0.7702	28.25/0.7721
BSD100	2×	32.02/0.8970	32.06/0.8978
	3×	28.93/0.7993	28.96/0.8011
	4×	27.32/0.7265	27.42/0.7286
Urban100	2×	31.04/0.9167	31.27/0.9191
	3×	27.39/0.8333	27.55/0.8384
	4×	25.36/0.7577	25.52/0.7649

D. Analysis

1) Depth Analysis: To analyze the influence of the network depth on the performance, we construct two networks with 9 and 25 blocks respectively and keep other parameters the same. Note that higher PSNR and SSIM values indicate better quality. As shown in Table. II, the model with 25 blocks (MODN-25B) demonstrates better reconstruction accuracy, which outperforms the model with 9 blocks (MODN-9B) more than 0.04 dB in PSNR and 0.001 in SSIM. This is mainly because the network with more parameters can extract more hierarchical features, which contributes to better performance. The depth analysis also indicates that our model allows deeper network for higher performance, so it is necessary to make it memory optimized for deeper structure.

2) Memory Analysis: To investigate the memory efficient property of the model, we compare the training-time memory consumption of the models with share memory allocation and the models without share memory allocation. For fair comparison, we use one 31×31 patch as input for training these models. As shown in Table. III, for model with 9 blocks depth, the memory optimized implementation (MODN-9B) consumes about 87% less memory than the 9 blocks model without share memory allocation (Plain-9B). For model with 25 blocks depth, the memory optimized implementation (MODN-25B) consumes 88% less memory than the 25 blocks model without share memory allocation (Plain-25B). This is

because MODN only stores convolution feature maps and network parameters in the memory. However, the plain model also needs to store concatenated and normalized features in the memory, in addition to the convolution feature maps and network parameters.

TABLE III
The Memory analysis of network

Model	Blocks	Memory(GB)
MODN	$9B$	0.2
	$25B$	0.8
Plain	$9B$	1.6
	$25B$	6.3

IV. Conclusion

In this work, we propose a memory optimized deep dense network for image super-resolution. First, we reduce the redundant feature learning by rationally implementing skip connection and dense connection in the network. Then, we construct share memory allocation for training the network to reduce the training-time memory. The memory optimized model requires less memory than normal deep dense model. We have demonstrated that our network outperforms the some state-of-art methods on the benchmark datasets with relatively less computation complexity.

References

[1] C. Dong, C. Loychen, H. Kaiming, and T. Xiaoou. "Learning a deep convolutional network for image super-resolution," European conference on computer vision. pp. 184-199, September 2014.

[2] K. Kwangin, and K. Younghee. "Learning a deep convolutional network for image super-resolution," IEEE Trans. Pattern Anal. Mach. Intell. vol. 32, pp. 1127–1133, 2010.

[3] K. Robert. "Cubic convolution interpolation for digital image processing," IEEE Trans. Signal Process. vol. 29, pp. 1153–1160, 1981.

[4] W. Shi, J. Caballero , and C. Ledig et al. "Cardiac image super-resolution with global correspondence using multi-atlas patchmatch," International Conference on Medical Image Computing and Computer-Assisted Intervention. Berlin. Heidelberg, pp. 9-16, 2013.

[5] Y. Tai, J. Yang, and X. Liu. "Image super-resolution via deep recursive residual network," IEEE Conference on Computer Vision and Pattern Recognition. 2017.

[6] K. Jiwon , K. Leejung, and M. Leekyoung. "Deeply-recursive convolutional network for image super-resolution," IEEE Conference on Computer Vision and Pattern Recognition. pp. 1637–1645. 2016.

[7] K. Jiwon , K. Leejung, and M. Leekyoung. "Accurate image super-resolution using very deep convolutional networks," IEEE Conference on Computer Vision and Pattern Recognition. pp. 1646–1654. 2016.

[8] H. Kaiming, Zhang. Xiangyu, Ren. Shaoqing, and S. Jian. "Deep residual learning for image recognition," IEEE conference on computer vision and pattern recognition. pp. 770–778. 2016.

[9] D. Chao, L. Chenchange, H. Kaiming, and T. Xiaoou. "Image super-resolution using deep convolutional networks," IEEE Trans. Pattern Anal. Mach. Intell. vol. 38, pp. 295–307, 2016.

[10] Z. Wang, C. BovikAlan, R.SheikhHamid, and P. SimoncelliEero. "Image quality assessment: from error visibility to structural similarity," IEEE Trans. Signal Process. vol. 13, pp. 600–612, 2004.

[11] S. Ioffe, and C. Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift," International Conference on Machine Learning. pp. 448–456, 2015.

[12] V. Nair, and E. HintonGeoffrey. "Rectified linear units improve restricted boltzmann machines," International conference on machine learning. pp. 807–814, 2010.

[13] Z. Yulun, T. Yapeng, Y. Kong, Z. Bineng, and F. Yun. "Residual dense network for image super-resolution," IEEE Conference on Computer Vision and Pattern Recognition. 2018.

[14] T. Tong, G. Li, X. Liu, and Q. Gao. "Image super-resolution using dense skip connections," IEEE International Conference on Computer Vision. 2017.

[15] W. W. Zou, and P. C. Yuen. "Very low resolution face recognition problem," IEEE Trans. Image Processing. 2012.

[16] D. Martin, C. Fowlkes, D. Tal, and J. Malik. "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," IEEE International Conference on Computer Vision. pp. 416–423, 2001.

[17] W. M. Thornton, M.P. Atkinson, and D. Holland. "Sub-pixel mapping of rural land cover objects from fine spatial resolution satellite sensor imagery using super-resolution pixel-swapping," International Journal of Remote Sensing. pp. 473–491, 2006.

[18] Y. Jianchao, J. Wright, T.S. Huang, and Y. Ma. "Image super-resolution via sparse representation," IEEE Trans. Image Processing. pp.2861–2873, 2012.

[19] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. "Densely Connected Convolutional Networks," IEEE Conference on Computer Vision and Pattern Recognition. 2017.

[20] M. Bevilacqua, A. Roumy, C. Guillemot, and M. Alberi-Morel. "Low-complexity single-image super-resolution based on nonnegative neighbor embedding," 2012.

[21] K. Malczewski, and R. Stasiski. "Super Resolution for Multimedia Image and Video Processing Applications,"Recent Advances in Multimedia Signal Processing and Communications. pp. 171-208.

[22] A. Small, and S. Stahlheber. "Fluorophore localization algorithms for super-resolution microscopy," pp. 267C279. 2014.

[23] S. Jian, X. Zongben, and S. Heung-Yeung. "Image super-resolution using gradient profile prior," Computer Vision and Pattern Recognition. 2008.

[24] A. Veit, J. Michael, and B. Serge. "Residual networks behave like ensembles of relatively shallow networks," Advances in Neural Information Processing Systems. 2016.

[25] T. Chen, B. Xu, C. Zhang, and C. Guestrin. "Training deep nets with sublinear memory cost,". 2016.

[26] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems,". 2015.

[27] P. Geoff, C. Danlu, H. Gao, L. Tongcheng, V. Laurens, and Q. Kilian. "Memory-efficient implementation of densenets,". 2017.

[28] J. Kim, J. K. Lee, and K. M. Lee. "Deeply-recursive convolutional network for image super-resolution,". 2016.

[29] D. Martin, C. Fowlkes, D. Tal, and J. Malik. "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics,". 2001.

[30] R. Timofte, V. De Smet, and L. Van Gool. "A+: Adjusted anchored neighborhood regression for fast super-resolution," Asian Conference on Computer Vision. pp.111–126. 2014.

[31] H. Jia-Bin, S. Abhishek, and A. Narendra. "Single image super-resolution from transformed self-exemplars," IEEE Conference on Computer Vision and Pattern Recognition. pp.5197–5206. 2015.