IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 5 October 2023, accepted 17 October 2023, date of publication 27 October 2023, date of current version 8 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3328173

RESEARCH ARTICLE

Accelerating Super-Resolution Network Inference via Sensitivity-Based Weight Sparsity Allocation

TUAN NGHIA NGUYEN[1], (Graduate Student Member, IEEE), XUAN TRUONG NGUYEN[2], (Member, IEEE), KYUJOONG LEE[3], (Member, IEEE), AND HYUK-JAE LEE[1], (Member, IEEE)

[1]Inter-University Semiconductor Research Center, Seoul National University, Seoul 08826, Republic of Korea
[2]Department of Next-Generation Semiconductor Convergence and Open Sharing Systems (COSS), Seoul National University, Seoul 08826, Republic of Korea
[3]School of AI Convergence, Sungshin Women's University, Seoul 02844, Republic of Korea

Corresponding author: Kyujoong Lee (kyujoonglee@sungshin.ac.kr)

This work was supported in part by the Technology Innovation Program (or Industrial Strategic Technology Development Program, Development of Technology for Commercializing Lv. 4 Self-Driving Computing Platform Based on Centralized Architecture) funded by the Ministry of Trade, Industry and Energy (MOTIE), South Korea, under Grant 20014490; and in part by the Sungshin Women's University Research Grant of 2023.

ABSTRACT Weight sparsification has been extensively studied in image classification and object detection to accelerate network inference. However, for image generation tasks, such as image super-resolution, forcing some weights to zeros is a non-trivial task that typically causes significant degradation in restoration quality, that is, peak signal-to-noise (PSNR). In this study, we first introduce a sensitivity metric to measure PSNR degradation for layer-wise sparsity changes and observe that the sensitivities vary significantly across network layers. We demonstrate that a uniform sparsity allocation method generally causes a non-negligible decrease in accuracy, that is, approximately 0.17 dB, for 65% of the zero weights. In addition, finding an optimal solution to the sparsity allocation problem is not feasible because the design space is exponential with respect to the number of weights and layers. To address this problem, we proposed a simple yet effective sparsity allocation method based on layer-wise sensitivity. The experimental results demonstrate that the proposed method achieves up to 35% computation reduction with an average accuracy drop of 0.02 dB varying between 0.01 to 0.04 dB across the well-known datasets Set5, Set14, B100, and Urban100. Moreover, when integrated with the activation sparsity SMSR, the proposed method reduced the computation by 46% on average.

INDEX TERMS Computational reduction, efficient neural network, sparsity.

I. INTRODUCTION

Transforming a low-resolution image into a high-resolution one, super-resolution (SR) has a wide range of applications such as improving medical image clarity [1], refining remote sensing imaginary for environmental monitoring [2], [3], and enhancing facial recognition systems [4]. The primary challenge in SR is to generate a realistic image from an image with limited detail. With the advent of convolutional neural networks (CNNs), solutions to this problem have consistently improved. One class of solutions utilizes subpixel adjustment on top of bicubic images to correct artifacts. To bridge the gap between theoretical and practical applications, the next step is to increase the processing speed of SR CNNs.

However, the processing speed of SR CNNs is restricted by their high computational costs. As the resolution demand for images continues to increase from high definition (HD) to full high definition (FHD), quad high definition (QHD), 2K, and 4K, the cost scales at a quadratic rate. For instance, scaling a 640 × 360 image to HD (at 1280 × 720) adds four times more information, whereas scaling the same image to QHD (at 2560 × 1440) adds 16 times more information. However, several details such as texture or smooth-colored regions can be duplicated from the corresponding pixels without additional calculations. This network can be designed to ignore components that are computationally unnecessary.

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Sharif.

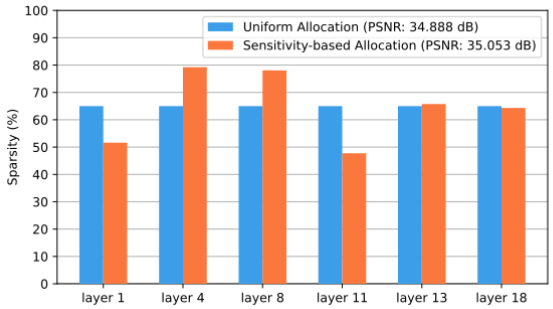

FIGURE 1. Sparsity allocation strategies on low, medium, and high-sensitive layers.

A computation involving zeros is potentially skippable, particularly multiplication, because it is predictable, for instance, multiplication with zero operands results in a zero value, regardless of the other operand. State-of-the-art architectures integrate a computational guide module to identify regions where the computations would result in zeros. Computation skipping can omit all or part of the computations of a feature, depending on the information provided by the guide. Nevertheless, the effort was reduced, resulting in improved overall processing speed.

Our work builds on the concept of skippable computation by exploiting the sparsity of model parameters, which represents the proportion of weights with zero values. We propose a simple yet effective method for increasing sparsity. Setting extremely small weights to zero increases the number of computations that can be ignored, ultimately enhancing the processing speed of the network. Performance degradation is unavoidable because of network modifications. To minimize this, we propose a sparsity allocation strategy based on a measurable sensitivity of each layer. As illustrated in Figure 1, while the uniform method applies the same amount for every layer, our sensitivity-based method suggests the exact sparsity that a layer should have. For instance, layers can be high (layers 1 and 11), medium (layers 13 and 18), or low (layers 4 and 8), depending on the extent to which the sparsification procedure affects the model performance, and sparsity is allocated accordingly. For the same target overall sparsity, the proposed method is better than the uniform solution, which does not consider the characteristics of each layer. Specifically, in terms of performance, the proposed model achieved 35.053 dB peak signal-to-noise (PSNR), which is higher than 34.888 dB on the DIV2K validation set. The visual difference can even be noticed in some cases as illustrated in Figure 2, where the detail of a sharp edge is well preserved by our proposed model while being mistakenly blurred out by another one. The proposed method incurs no overhead during inference because weight sparsification is performed during the training phase.

In the next section, several related topics are reviewed. Our proposed method and associated problems are discussed in Section III, along with our solutions. Experimental setup and results are presented in Section IV to support our claims. Finally, conclusions are presented in the last section.

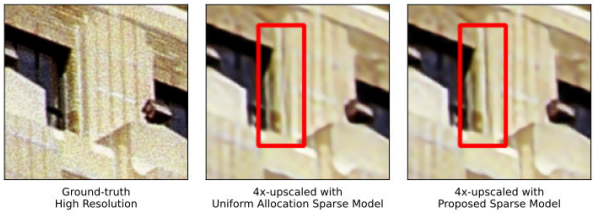

FIGURE 2. Visual difference between upscaled images generated by sparse models using uniform allocation and sensitivity-based allocation.

II. RELATED WORK

In this section, we review major studies on efficient SR networks and acceleration techniques.

A. GENERAL AND EFFICIENT SR NEURAL NETWORKS

SR neural networks are used to increase the resolution of an image and are useful in various applications such as image processing, video compression, and computer vision. The VDSR [5] was one of the first high-performance SR networks introduced. It utilizes an extremely deep network to achieve state-of-the-art results. The enhanced version, EDSR [6], improves image quality by increasing the depth of the network. The authors showed that increasing the depth of the network further improved the performance. Following this trend, HDN [7] uses very dense connections to promote the feature representation and RDN [8] goes beyond 100 layers with residual and dense connections. The attention mechanism was also applied to SR with the introduction of a very deep network, RCAN [9], followed by ENLCA [10]. ESRT [11] harnessed the power of Transformer [12] architectures, originally developed for natural language processing, for the task of single image SR. Several papers are leaning towards the application of SR on satellite imagery. EEGAN [13], which employs the popular Generative Adversarial Network (GAN) [14] technique, and D2U [15], which incorporated contrastive learning, are two notable works proposed to improve low-resolution remote sensing images.

Considering the exploding computational cost, a branch of SR network design focuses on the tradeoff between performance and efficiency. The early FSRCNN [16] modified the original SRCNN [17] to speed up the network 40 times while also improving the output quality. LapSRN [18] increases the computational cost but improves the PSNR by 0.5 dB which is significant for SR tasks. Recently, CARN and CARN-M [19] proposed a cascading architecture to utilize multilevel features to achieve efficiency across multiple scaling factors. There are a few other well-known names for lightweight SR networks such as DRCN [20], MemNet [21], SRFBN-S [22], IDN [23], and FALSR-A [24]. All the previously mentioned networks utilize their unique architectures to achieve a good balance between performance and computational efficiency.

Our work is related to the recent design of tiny and low-cost neural networks [25], [26], [27], [28]. These studies propose methods that have the potential to enhance the application

of lightweight neural networks. By leveraging the sparsity of neural networks, our approach enhances the accuracy and efficiency of these networks. Therefore, our study represents an important step towards advancing the state-of-the-art in computation reduction.

B. DYNAMIC INFERENCE

By switching between shallow and deep subnetworks, the computational cost depends on the characteristics of the input instances and is reduced on average. BlockDrop [30] and Branchynet [31] addressed instance-specific difficulties and introduced dynamic network architectures. Instances were analyzed by a controller in the network and fed through a specific path formed by a set of pre-trained modules. Adaptive architectures also exist for SR tasks. The spatially adaptive computation time (SACT) mechanism [32] stops computing when the features are sufficiently accurate. Dynamic inference requires careful design choices owing to the additional overhead incurred.

C. SPARSITY EXPLOITATION

Because of the large input and output sizes, the computational cost of an SR neural network is significant. Exploiting sparsity (i.e., zero-value operands and features), a highly sparse feature map is a frequent result of using ReLU activation because the feature is zero if its intermediate value computed via a multiply-accumulate operation is non-positive. Based on this observation, CSM [33] proposed using an additional module to predict the sparsity of the output feature maps and identify those to compute, thereby effectively reducing costs. SMSR [34] shares the same idea, but with a better mask-generation scheme that further improves efficiency. This is based on the fact that not all regions of an image require the same level of detail and that the architecture can be far more efficient by focusing on some important regions. SMSR uses a two-stage architecture. The first stage generates a sparse mask that indicates the regions of the image that require computing. The second stage is an SR network that uses sparse masks to selectively perform computations on the features of these regions. This reduces the overall computational cost while maintaining high-quality results. The results of the experiments demonstrate that SMSR outperforms state-of-the-art methods on various datasets and scales while reducing the computational cost by up to 50%.

D. PRUNING AND QUANTIZATION

In addition to directly incorporating computation reduction enhancements in architectures, general techniques that can be applied to general networks are also considered in SR. Quantization and pruning are popular techniques used for this purpose. Pruning directly modifies architecture by removing unimportant parameters. However, quantization reduces the model size and enhances the processing speed of the model on hardware. Weight sparsification is a general technique complementing these methods. We targeted unimportant parameters and ignored the corresponding computations. Unlike pruning, there is no modification of the architecture, only to the parameters. A pruned and/or quantized model can be sparsified to further enhance inference speed.

III. PROPOSED METHOD

A. WEIGHT SPARSIFICATION

The parameters of a layer, also known as weights $\mathbf{W}$, participate in the computation of the output features. The multiplication of one weight and the input feature contributed to the feature value. However, because an output feature is computed as an accumulation of several of these results, weight values close to zero contribute little to the final value. In other words, there is no or minimal alteration to the feature being computed even if an approximately zero-multiplication result is treated as exactly zero. In this case, the entire procedure can be safely omitted.

We propose weight sparsification as a technique that forces the value of a number to zero, with negligible performance degradation. The basic idea is that a threshold is selected for each set of weights, and all weights with absolute values lower than the threshold are rounded to zero. The sparsification process follows Equation 1, where w_i^l denotes i^{th} parameter of layer l and t^l denotes the threshold. We propose layer-wise weight sparsification in which the parameters of the same layer share a single threshold.

$$w_i^l = \begin{cases} w_i^l & \text{if } |w_i^l| \geq t^l \\ 0 & \text{if } |w_i^l| < t^l \end{cases} \tag{1}$$

We formulated the optimization problem by maximizing the model performance (or minimizing the performance degradation) while achieving the target sparsity S using a weight sparsification technique. The model performance $\mathbf{P}$ is a function of the layer weights $\mathbf{W}^l$ and sparsification threshold, which is indirectly represented by index value k_i. The constraint of this objective function was the overall sparsity of the model.

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & \mathbf{P} = f(\mathbf{W}, k_1, \ldots, k_M) \\ \text{subject to} \quad & \frac{\sum k_i}{M \times N} = S, \ i = 1, \ldots, M. \end{aligned} \tag{2}$$

The number of weights is finite and static during the inference. Therefore, according to Eq. 1, the threshold can be selected from the absolute weight values. This reduces the number of candidates and search time for an optimal threshold. However, with M target layers and N weights per layer, there are still N^M candidate solutions. The size of candidates is considered to be significant. A heuristic algorithm is necessary to efficiently explore the design space.

B. SENSITIVITY MEASUREMENT

The selection of a threshold for a layer forces a number of weights to zeros and increases the sparsity of that layer

Algorithm 1 Measuring Sensitivity of a Layer

Data: Layer weights $\mathbf{W}$ with N elements, training data $\mathbf{D}$

1 We set the overall performance boundary $\mathbf{P}_b$
2 $\bar{\mathbf{W}} = \text{sorted}(|\mathbf{W}|) = \{\bar{w}_k\}$ // sorted in ascending order of absolute values, where the index $k \in [0, N-1]$
3 $k_l = 0$ // left boundary
4 $k_r = N - 1$ // right boundary
5 $k_c = (k_l + k_r)/2$
6 $k_{best} = 0$
7 **while** $k_l \neq k_r$ **do**
  // binary search for threshold
8 $t = \bar{w}_{k_c}$ // the threshold value is the k_c-index element of the sorted weights
9 Sparsify the weights $\mathbf{W}$ with threshold t
10 Check performance $\mathbf{P}$ of the new model for $\mathbf{D}$
11 **if** $\mathbf{P} < \mathbf{P}_b$ **then**
12 $k_r = k_c$
13 **end**
14 **else**
15 $k_l = k_c$
16 $k_{best} = k_c$
17 **end**
18 $k_c = (k_l + k_r)/2$
19 **end**

Result: k_{best}

(parameters). To achieve the target sparsity S, this act allocates sparsity from the budget to the aforementioned layer. An optimal allocation to every target layer is required for our problem. One of the simplest allocation strategies is to uniformly sparsify every layer with the same sparsity S. However, in this study, we argue and prove that uniform allocation is a naive approach that completely ignores characteristic differences among layers, and thus degrades the overall performance non-negligibly.

We used a metric known as sensitivity to represent the characteristics and allocate sparsity to a layer. Sensitivity is inversely proportional to the maximum percentage of weights that can be sparsified without lowering performance. It is possible to force more weights to zeros with a low-sensitivity layer, and vice versa. For instance, we can sparsify 20% of the weights of layer 1 and obtain a model with 35 dB PSNR for the SR task. Simultaneously, 40% of the weights of layer 2 can be sparsified and we also obtain a model with 35 dB PSNR. Therefore, we can assume that Layer 1 is more sensitive than Layer 2. In other words, the model performance was significantly affected when its highly sensitive layers were modified. The determination of an appropriate threshold was based on the sensitivity of each layer.

Algorithm 1 describes sensitivity measurement for a single layer. First, we set a performance boundary $\mathbf{P}_b$ (line 1), which was measured using the PSNR. Performance degradation is expected to occur after sparsification, which is a post-training modification of model parameters. The algorithm determines the highest possible sparsity that can be achieved without lowering the performance beyond this boundary. In the next step, the absolute values of the weights $\mathbf{W}$ are sorted in ascending order as $\bar{\mathbf{W}} = \{\bar{w}_k\}$. As discussed, the threshold was selected from among all absolute weight values (line 2). Because the number of parameters N is often considerably large (thousands per layer), linear searching for the threshold is as expensive as making N model inferences on the training set $\mathbf{D}$, which is $\mathcal{O}(N)$. Therefore, a binary search was applied, which significantly reduced the time complexity to $\mathcal{O}(\log N)$. The search begins by setting the left and right bounds of the binary search (lines 3 and 4) and calculating the search point at the center of the bounds (line 5). The best threshold was initialized as $k_{best} = 0$. The primary loop of the binary search begins by setting a threshold for the sparsification process. As mentioned previously, the threshold value t is equal to the k_c-indexed value of the sorted weights (line 8). We sparsify the original model with t and check the performance of the new model on dataset $\mathbf{D}$. With a low search duration, owing to the binary search, we can use sufficiently large training data $\mathbf{D}$ to avoid noisy and biased performance on several samples. If the performance $\mathbf{P}$ of the sparse model is lower than the boundary, then the right bound is reduced to k_c (lines 11–13). Otherwise, we increase the left bound to k_c and record the new best threshold (lines 14–17): The output of Algorithm 1 is k_{best}, indicating that, at most, k_{best} of N parameters can be sparsified to achieve a performance no lower than the boundary $\mathbf{P}_b$. The equivalent sparsity converted from the value k_{best} is $s = k_{best}/N$.

Algorithm 1 is applied to each target layer. To compare the sensitivity of each layer fairly, the same performance boundary was used. Searching for the layer that has a greater impact on the overall performance once modified is the first important step in our sparsity allocation strategy. The threshold is thereafter determined using this metric.

C. ALLOCATION OF SPARSITY

Because each layer has its own sensitivity, it is preferable for each layer to have its own sparsity level. The target sparsity is distributed strategically among layers such that highly sensitive layers have slightly modified weights, whereas the others have much sparser weights. To achieve S, the number of parameters to be sparsified is $S \times M \times N$, assuming that we target M layers having N parameters each. The allocation strategy follows the rule described in 3. In this equation, for each layer i, the number of parameters to be sparsified, k_i', scales linearly with the sensitivity, represented by k_i. Therefore, the calculation of k_i' can be derived from Equation 3 to Equation 4 because the budget of sparse weights is divided into M parts by the allocation ratio $k_i / \sum k_i$. Using this formula, the thresholds and allocations to each layer can

be calculated directly rather than using a greedy search.

$$\frac{k_i'}{S \times M \times N} = \frac{k_i}{\sum k_i} \tag{3}$$

$$k_i' = S \times M \times N \times \frac{k_i}{\sum k_i} \tag{4}$$

D. INTEGRATE WEIGHT SPARSIFICATION INTO SMSR

Weight sparsification alone reduces the number of computations based on the amount of target sparsity, offering a significant advantage over the inference speed of SR networks. In this section, we present the integration of this technique into SMSR [34] to prove that it complements other state-of-the-art techniques.

The original version of SMSR has computation guide modules known as sparse mask modules (SMMs) integrated into its architecture. The modules compute several sparse masks that contain the sparsity information of all feature maps computed in the network. Knowing the features that are sparse allows the network to ignore all the corresponding computations. We refer to this technique as output-skipping. The second is input skipping, which aims to skip all multiplications with zero-value input features. Input skipping is similar to exploiting sparse weights for computational reduction because the input feature and weight are the two operands of an operation. Our sparsification method makes the weights sparse, thus enabling their exploitation to skip computations.

IV. EVALUATION

A. EXPERIMENT SETTINGS

1) TARGET NETWORK

Enhanced versions of the SMSR were used in the experiments. We used weight sparsification to further reduce the computational cost of this network, proving the benefit offered by the proposed method as well as its complement to other state-of-the-art techniques. The proposed method was applied to five SMMs of an SMSR. There are $M = 20$ layers to be considered and each layer has $N = 36864$ parameters ($3 \times 3 \times 64 \times 64$). For fairness, the original published 2x-scaling model was used. In the case of the 4x-scaling one, we retrained the model using the exact settings presented in the original paper and verified its performance with the published results provided by the authors. All the implementations were performed using PyTorch [35]. All model modifications were performed without additional retraining or fine-tuning phases.

2) DATASET

The SMSR is trained using the DIV2K dataset [36]. A total of 800 images were used to train, and 100 images were used for validation. The PSNR and structural similarity index measure (SSIM) were used as the SR performance metrics. The evaluation was performed on the luminance channel (Y channel) of each image with cropped borders. The DIV2K valid set was used to monitor the performance of the network during weight sparsification. The final results were tested using Set5 [37], Set14 [38], B100 [39], and Urban100 [40], which are popular test sets for SR tasks.

TABLE 1. Measurement of sensitivity for 20 core layers of SMSR.

Layer / ΔP	0.01	0.02	0.03	0.04
1	34.52	42.93	49.70	52.56
2	38.41	53.15	58.38	62.83
3	49.12	58.75	62.25	67.61
4	60.54	67.11	73.98	80.66
5	36.27	44.94	55.37	59.70
6	35.74	47.28	50.68	57.06
7	50.12	59.38	71.12	75.21
8	55.65	67.10	74.59	79.48
9	42.52	51.26	55.88	61.33
10	37.67	46.18	52.16	56.72
11	38.10	44.85	47.62	48.60
12	43.14	49.08	52.13	54.53
13	43.79	53.33	61.59	66.94
14	52.50	61.36	67.52	70.55
15	52.42	60.71	70.84	75.98
16	58.57	69.96	74.93	77.38
17	40.75	52.55	57.57	61.75
18	43.63	53.54	57.89	65.49
19	48.32	61.67	67.07	71.02
20	58.60	68.91	73.81	78.14

3) TRAINING SERVER

The experiments were conducted on a server with an Intel Xeon E3-1245 v5 CPU, two NVIDIA GPUs, Titan X and RTX 2080Ti, and 32 GBs of memory. It took up to four hours to measure the sensitivity of 20 layers. The sparse model can thereafter be generated instantly and reused during inference without incurring any overhead.

B. ANALYZE WEIGHT SPARSIFICATION

Table 1 lists the sensitivity of each layer, represented by the maximum sparsity (in percentage) achieved without degrading the performance by more than ΔP. The table can be interpreted such that, for instance, at $\Delta P = 0.04$, 80.66% of the weights of Layer 4 can be forced to zeros, whereas only 48.60% of Layer 11 can be sparsified without causing 0.04 dB degradation of the overall performance. This indicates that Layer 11 is more sensitive than Layer 4. The sensitivity measured at this boundary was used to calculate the allocation ratio.

The proposed method was compared to uniform sparsity allocation. The uniform allocation method sparsifies the weights of each layer to the same level as the overall target sparsity S. The results are summarized in Table 2. We calculated the PSNR changes between the models before and after sparsification. A model is better than the others if it has lower degradation. It can be observed that at the same target sparsity, the proposed method yields better performance than the others most of the time. At 5% overall sparsity, the sparse model had an approximately similar PSNR to the original one. The degradation is typically 0.002—0.165 dB lower when using our method than uniform allocation. At 35% target sparsity, the degradation is approximately 0.1 dB,

TABLE 2. Sensitivity-based allocation vs. uniform allocation. The metric is PSNR (perturbation, in dB). Lower degradation is better.

Target Weight Sparsity (%)	Uniform Allocation	Sensitivity-based Allocation
5	0.001	**0.000**
10	0.003	**0.002**
15	0.009	**0.007**
20	**0.016**	0.018
25	0.034	**0.031**
30	0.072	**0.058**
35	0.125	**0.101**
40	0.204	**0.155**
45	0.307	**0.221**
50	0.450	**0.352**
55	0.662	**0.485**
60	0.799	**0.631**
65	0.971	**0.805**
70	1.196	**1.050**
75	1.356	**1.269**

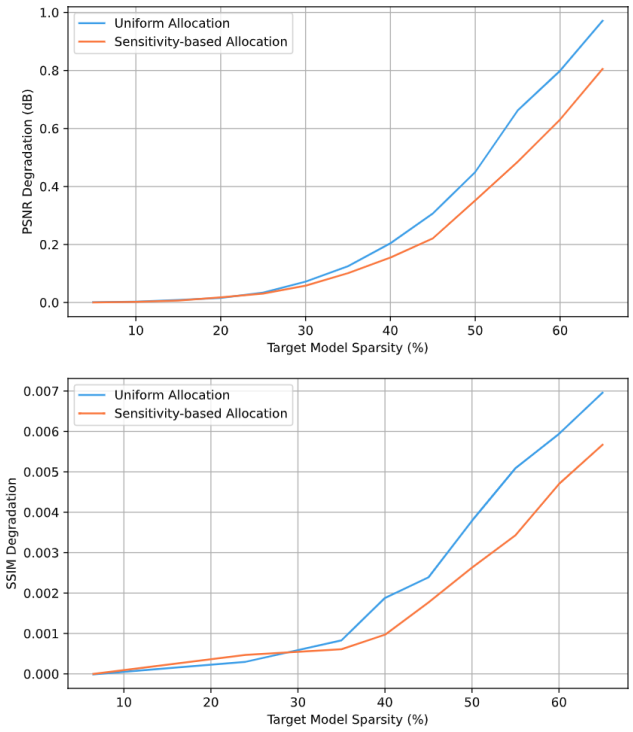

FIGURE 3. Trade-off between sparsity and PSNR/SSIM degradation. Lower degradation is better.

which is considered non-negligible. Beyond this point, the model collapses exponentially, as illustrated in Figure 3. Therefore, we can conclude that the proposed method can ignore up to 35% of the weights with negligible degradation, which is equivalent to 35% of the computations in the target layers.

C. SPARER SMSR (SPARSMSR)

A comparison of different versions of SMSR is summarized in Table 3. Three techniques, namely output skipping, input skipping, and weight sparsification, are considered to reduce the computational cost of the target layer. The number of computations in Giga floating-points operations (GFLOPs) can be calculated based on the network settings, including the layer filter and input sizes. Specifically, the input size was 640 × 360 for the 2x-scaling models and 320 × 180 for the 4x-scaling ones. The total cost is considered for each layer. We also calculated the computational reduction benefit for only the 20 main layers following the design of SMSR. Without the SMMs, the total computational cost was approximately 212.44 GFLOPs to scale 2× and generate a high-definition image (1280 × 720). Applying output skipping and input skipping saves 53.4% of the computations in the 20 layers, and this model has a total cost of 133.03 GFLOPs. The Sparser SMSR (SparSMSR) 1a model, which has weights sparsified by 6.5%, can perform a similar task; however, the reduction rate is slightly lower at 52.7%. Therefore, sparsifying weights offers fewer benefits. However, skipping computations that use sparse weights can be applied in conjunction with an input-skipping scheme of SMSR. This was proven using the SparSMSR 1b model. This version applies all three techniques, and the rate increases to 56.4% without sacrificing the performance. Each technique, input skipping and exploiting sparse weights, offers fairly high benefits if used alone. However, the benefits do not simply add up because multiplication with both zero weight and zero input as operands is ignored in the same way as multiplication with either weight or input being zero. This explains why the improvement was minor (from 53.4% to 56.4%) when the two were combined.

When a minimal performance degradation of up to 0.1 dB is acceptable, we can consider four other versions, SparSMSR 2a, 2b, 3a, and 3b, with higher weight sparsity. At 24% and 35% sparse weights, we ignored 13.7% and 21.0% more computations, respectively, compared with the original SMSR. Exploiting sparse weights is a significant advantage because no overhead is incurred during inference. This implies that our method can provide additional benefits at a minimal cost. In the case of the SparSMSR 1a and 1b models, no performance degradation was observed.

Extensive experimental results on 4x-scaling models also prove the benefits of weight sparsification on the computation cost. Our enhanced models, SparSMSR 4 × 1b, 2b, and 3b achieved 39.4%, 50.7%, and 58.4% computation reduction in the target layers, respectively. Considering the entire model, these numbers can be converted to 25.6%, 34.4%, and 40.5% of the total reduction in GFLOPs, respectively. Of the three proposed models, SparSMSR 4 × 2b performs remarkably well with almost no degradation to the PNSRs on all four test sets.

The allocation of sparsity for 2x-scaling models can be found in Figure 4 with high-sensitive layers sparsified at a lower rate than those with lower sensitivities. For instance, to achieve 35% overall weight sparsity, we sparsify Layer 4 at 42.6% and Layer 11 at 25.70% because their sensitivities were 80.66 and 48.60%, respectively. The ratios were approximately equal at 42.66 / 80.66 and 25.70 / 48.60. The allocation difference between the layers becomes evident

TABLE 3. Comparison between the original and enhanced versions of SMSR with weight sparsification.

Model	Weight sparsity %	Output skip	Input skip	Computation reduction %	Computation total GFLOPs	PSNR / SSIM Set14	Set5	B100	Urban100
SMSR w/o SMM [34]	0.0			0.0	212.44	33.64/0.9179	38.00/0.9601	32.17/0.8990	32.19/0.9284
SMSR [34]	0.0	✓	✓	53.4	133.03	33.62/0.9177	37.99/0.9601	32.15/0.8990	32.19/0.9284
SparSMSR 1a	6.5	✓		52.7	134.21	33.62/0.9177	37.99/0.9601	32.16/0.8990	32.19/0.9284
SparSMSR 1b	6.5	✓	✓	56.4	128.00	33.62/0.9177	37.99/0.9601	32.16/0.8990	32.19/0.9284
SparSMSR 2a	24.0	✓		61.3	119.67	33.60/0.9172	37.98/0.9601	32.14/0.8988	32.15/0.9278
SparSMSR 2b	24.0	✓	✓	64.2	114.76	33.60/0.9172	37.98/0.9601	32.14/0.8988	32.15/0.9278
SparSMSR 3a	35.0	✓		67.5	109.21	33.55/0.9171	37.94/0.9600	32.09/0.8982	31.95/0.9261
SparSMSR 3b	35.0	✓	✓	69.9	105.05	33.55/0.9171	37.94/0.9600	32.09/0.8982	31.95/0.9261
SMSR 4x w/o SMM [34]	0.0			0.0	54.30	28.55/0.7808	32.12/0.8932	27.55/0.7351	26.11/0.7868
SMSR 4x [34]	0.0	✓	✓	35.3	42.15	28.55/0.7812	32.05/0.8931	27.54/0.7346	26.09/0.7859
SparSMSR 4x 1b	6.5	✓	✓	39.4	40.40	28.55/0.7812	32.05/0.8932	27.54/0.7346	26.09/0.7859
SparSMSR 4x 2b	24.0	✓	✓	50.7	35.64	28.54/0.7808	32.05/0.8928	27.53/0.7342	26.06/0.7848
SparSMSR 4x 3b	35.0	✓	✓	58.4	32.33	28.51/0.7801	32.05/0.8927	27.50/0.7337	25.97/0.7825

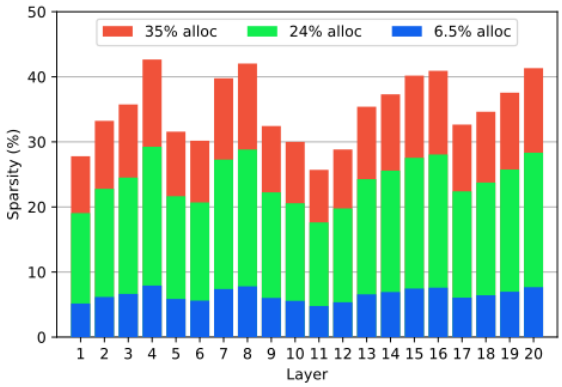

FIGURE 4. Allocation of sparsity to each layer following a ratio calculated by the sensitivities.

TABLE 4. Comparative results achieved on Set14 by SparSMSR and other architectures for 2× SR.

Model	Non-zero Params	FLOPs	CPU Time	PSNR	SSIM
SMSR w/o SMM [34]	926K	1.00 ×	1.00 ×	33.64	0.9180
SMSR [34]	985K	0.63 ×	0.79 ×	33.62	0.9179
SparSMSR 2b	808K	0.54 ×	0.78 ×	33.60	0.9172
SparSMSR 3b	727K	0.49 ×	0.68 ×	33.55	0.9171
IDN [23]	533K	0.57 ×	0.73 ×	33.30	0.9148
CARN [19]	1592K	0.99 ×	0.89 ×	33.52	0.9166
FALSR-A [24]	1021K	1.04 ×	1.05 ×	33.55	0.9168

as the target sparsity increases. As summarized in Table 2, the PSNR improvement achieved with our proposed method is clearly higher than that of uniform allocation when the model is heavily modified via sparsification.

The measurement of inference time for various models on a CPU is presented in Table 4, proving the positive impact of sparse weights on inference efficiency. Notably, the incorporation of sparse weights unlocks the potential for an efficient use of loop unrolling. Even without optimizing the implementation of sparse inference, the results reveal that SparSMSR 2b and 3b still manifest a substantial reduction in inference time, with 22% and 32% improvements, respectively, thanks to that only 808K and 727K parameters out of 985K are involved in all the computations. Remarkably, when considering comparable levels of performance in PSNR and SSIM, SparSMSR 3b emerges as superior in comparison to the state-of-the-art efficient SR models such as FALSR-A [24], CARN [19], and IDN [23].

While it is worth noting that the relative reduction in floating-point operations may not directly align with the same percentage of time saved, it nevertheless presents a significant advantage in terms of power consumption. This aspect bears particular relevance for the implementation of these models in edge devices, where power efficiency is a paramount consideration. In essence, our findings highlight the potential of sparse weight structures to enhance inference efficiency, offering both performance gains and energy-saving benefits that hold promise for practical deployment in resource-constrained environments.

D. USING SSIM IN SENSITIVITY MEASUREMENT

SSIM can be used to measure the sensitivity of each layer in the same way as using PSNR. We set the performance degradation threshold in Algorithm 1 to 0.0005 due to that the change of SSIM is usually more sensitive to the image quality than that of PSNR. Measuring with SSIM results in a slightly different allocation. The results are summarized in Table 3, and they are aligned with the ones obtained when PSNR was used. It showed that we have successfully reduced more computations at a negligible SSIM loss. In addition, a comparison between uniform allocation and the proposed method is visualized in Figure 3. In general, it is proven that our strategic allocation offers a better preservation of performance under a harsh modification of the model parameters.

E. SPARSIFICATION OF VERY DEEP NETWORKS

We set out to explore the plug-and-play applicability of our proposed framework in the context of very deep neural networks, with a particular focus on the extensive RCAN architecture [9]. To this end, we conducted an in-depth investigation involving 400 layers within the RCAN model. The findings of this comprehensive sparsification process

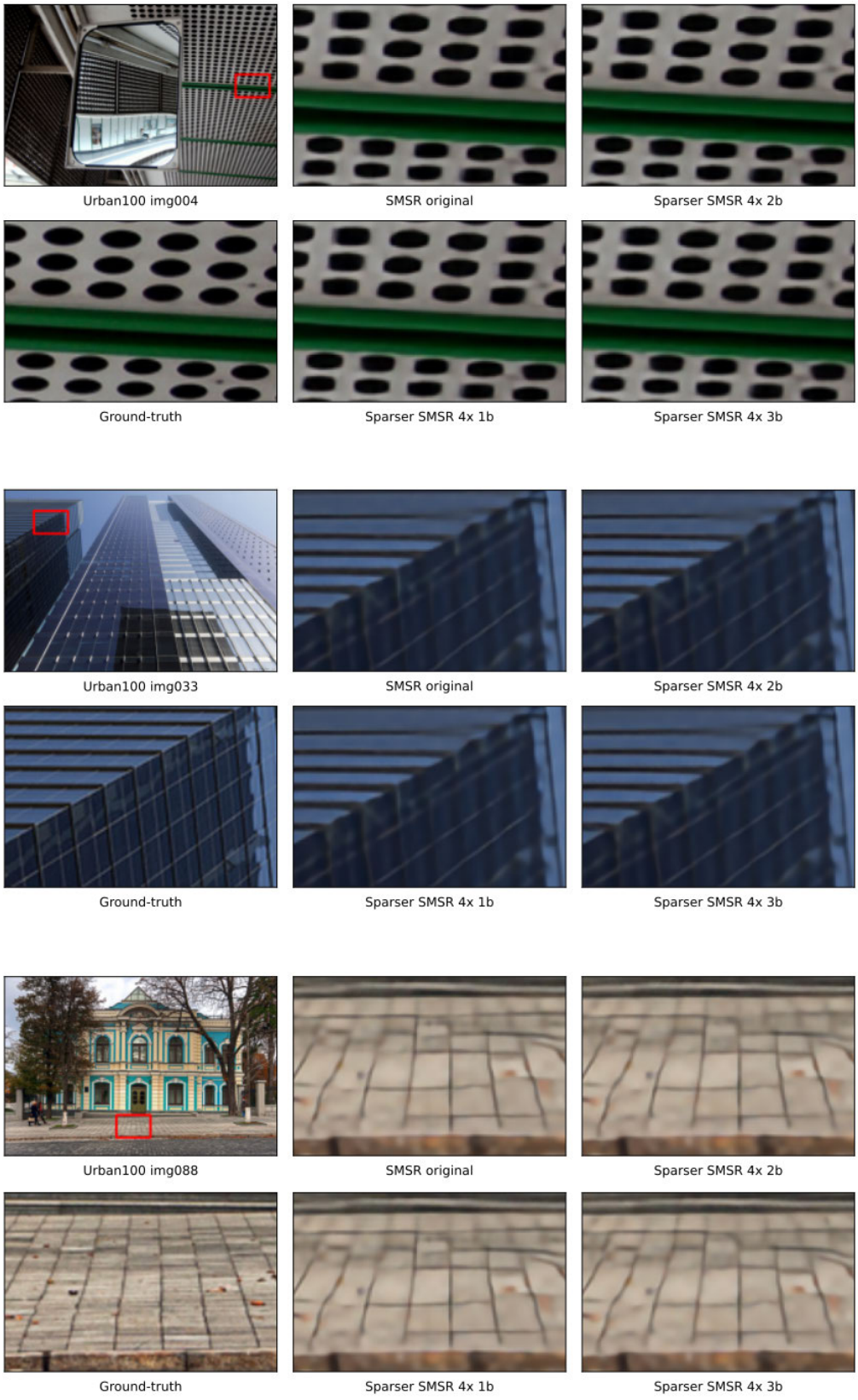

FIGURE 5. Visual comparison on three sample images of the Urban100 dataset. SR results are generated by four different versions of SMSR.

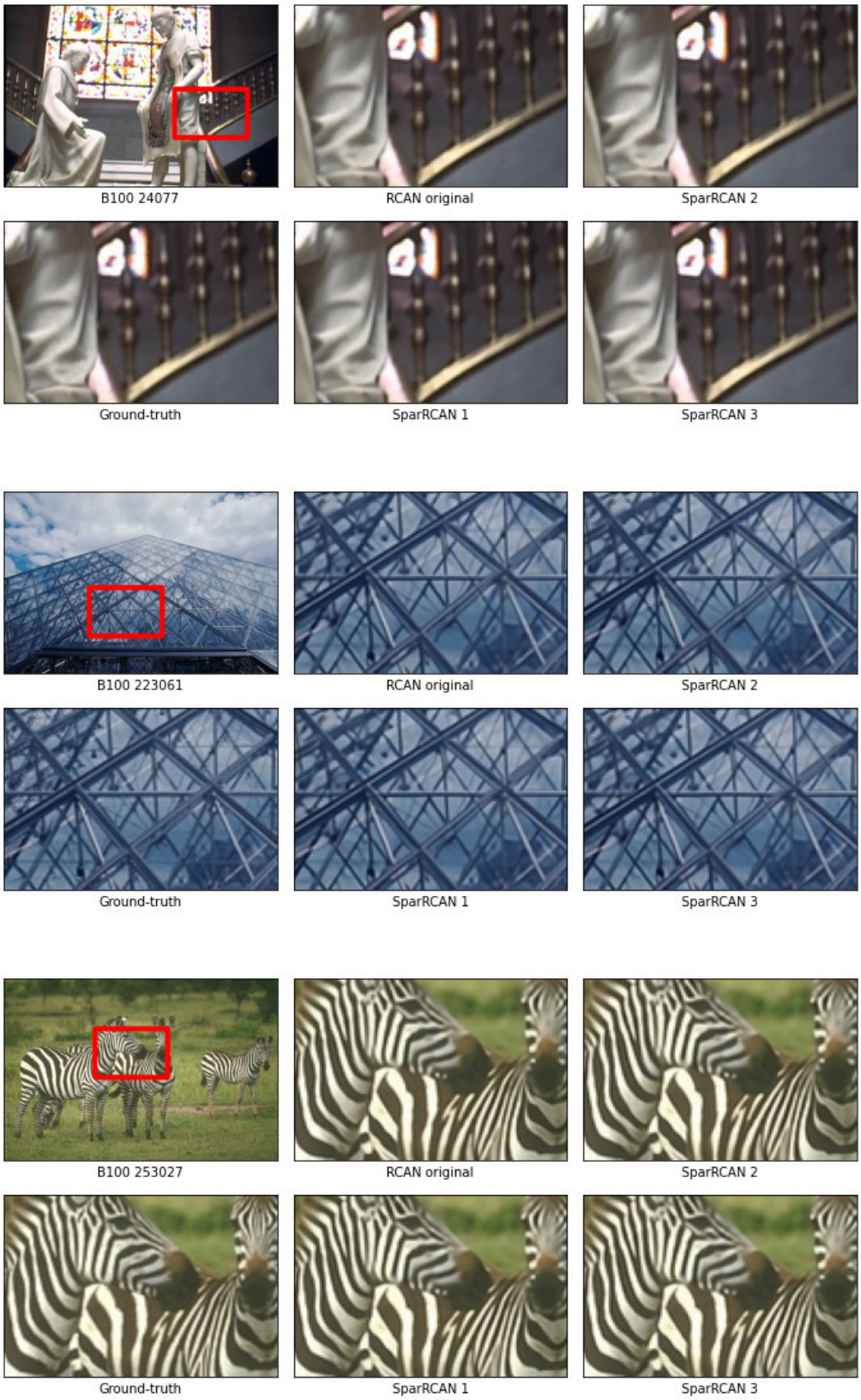

FIGURE 6. Visual comparison on three sample images of the B100 dataset. SR results are generated by four different versions of RCAN.

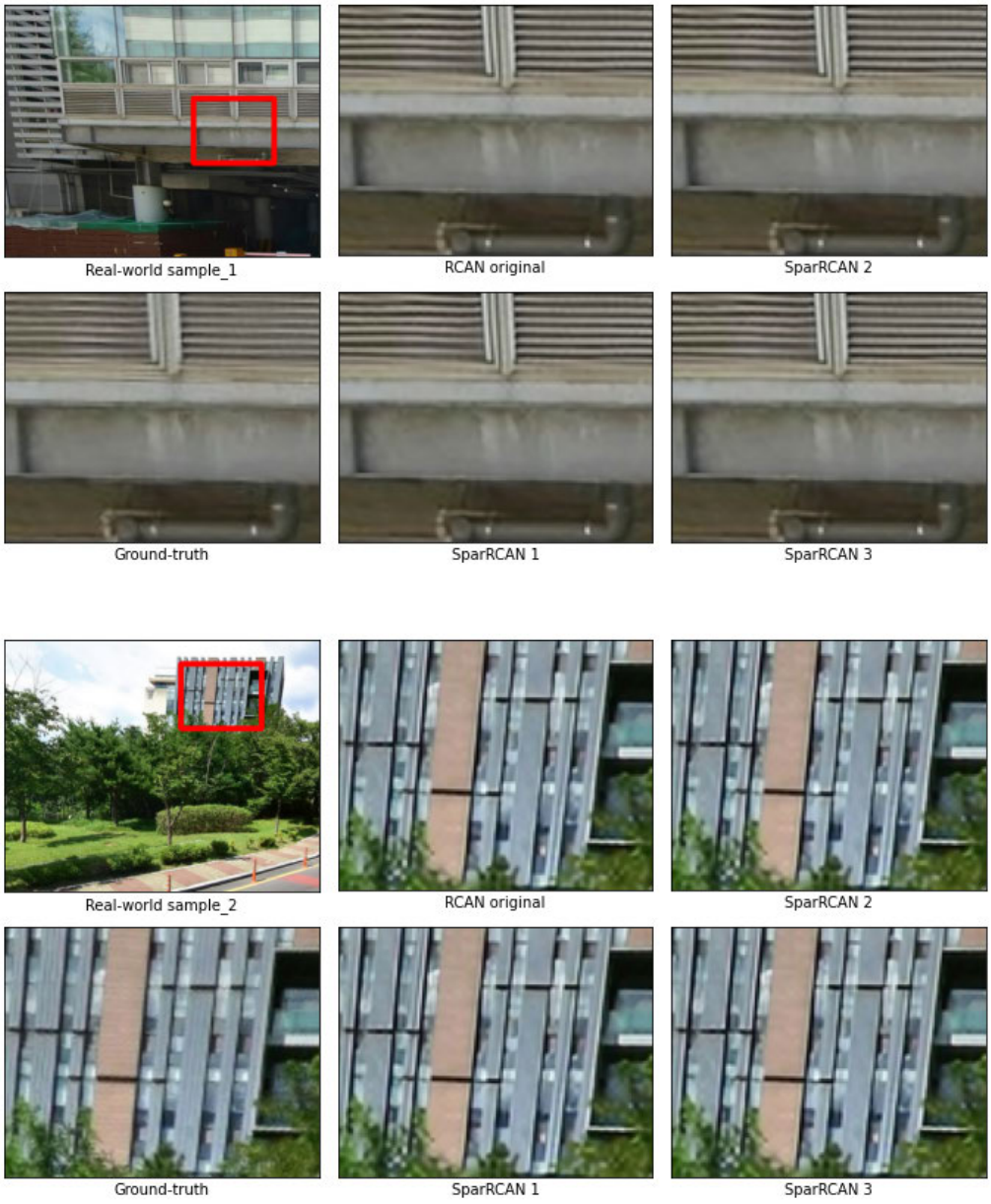

FIGURE 7. Visual comparison on two real-world sample images.

TABLE 5. Comparison between the original and enhanced versions of RCAN with weight sparsification.

Model	Weight sparsity (%)	Non-zero Params	FLOPs	PSNR / SSIM			
				Set14	Set5	B100	Urban100
RCAN [9]	0.0	15.4M	1.00 ×	34.12/0.9214	38.27/0.9613	32.39/0.9022	33.18/0.9370
SparRCAN 1	10.0	13.9M	0.90 ×	34.12/0.9214	38.27/0.9613	32.39/0.9022	33.18/0.9370
SparRCAN 2	25.0	11.7M	0.76 ×	34.08/0.9211	38.25/0.9612	32.38/0.9021	33.13/0.9366
SparRCAN 3	35.0	10.2M	0.66 ×	34.05/0.9211	38.22/0.9611	32.36/0.9018	33.05/0.9360

have been compiled in Table 5, offering a clear demonstration of the potential our approach holds.

These 400 layers, which constitute a substantial 96% of the overall computational load in the RCAN network, emphasize a crucial aspect of our work: the sparsity of weights within these layers leads to almost proportional reductions in computational requirements. To illustrate this, take the case of SparRCAN 1, where we successfully achieved a 10.0% weight sparsity. This accomplishment directly translated into a 10.0% decrease in computational workload, a trend that was consistently observed across SparRCAN 2 and 3 (24% and 34%, respectively). Sparsification helps reduce significantly the number of non-zero parameters from 15.4M to 13.9M, 11.7M, and 10.2M in the three proposed models. In other

words, all the computations in which the zero-value parameters involved are skipped. Such results prove the effectiveness of our framework in enhancing the computational efficiency of very deep neural networks.

Shifting our attention to the performance aspect, it is worth noting that the decline in PSNR remains virtually negligible until the weight sparsity reaches the 35% threshold. While this sparsity level might typically raise concerns about performance in conventional settings, it is essential to consider the substantial reduction in computational demands that accompanies it. This trade-off, therefore, emerges as not only reasonable but highly advantageous in scenarios where computational efficiency is a paramount consideration. In summary, our findings underscore the adaptability of our proposed framework and provide valuable insights into the complex relationship between weight sparsity, computational efficiency, and performance in deep neural networks, with implications for the broader field of deep learning and optimization.

F. QUALITATIVE RESULTS

In addition to the quantitative results demonstrated by the PSNR, Figure 5 and Figure 6 illustrate four different upscaled versions of three sample images from the Urban100 dataset and the B100 dataset respectively. The images in Figure 5 were generated using the original SMSR model and three sparse 4x-scaling models, 1b, 2b, and 3b. Remarkably, there were hardly any perceptible differences among these versions. A similar scenario can be observed in Figure 6. Moreover, for a diverse qualitative assessment, we present two upscaled real-world samples, as depicted in Figure 7. It can be inferred that the marginal variance in PSNR did not yield any visual disparities, thus preserving the output image quality effectively. Consequently, it is reasonable to adopt the proposed method, as it offers significant advantages at minimal cost.

V. CONCLUSION

This study presents the use of sparse weights to reduce the computational costs of neural networks. Small-value weights were forced to zeros, and all computations involving zero weights were ignored during the inference. A strategy to measure the sensitivity and allocate sparsity to each layer was presented to minimize the negative impact on model performance. The experiments on a state-of-the-art efficient SR model demonstrate that with all three techniques applied, we can reduce the overall computational cost from 212.44 to 105.05 GFLOPs, which is equivalent to 50.6%, at a cost of only 0.1 dB lower PSNR and no visually noticeable quality change to the output images. This technique is simple and applicable to various SR neural networks, yet powerful for deploying a model on hardware and maximizing its benefits.

A significant limitation of our current work lies in its data dependency. To effectively model the sensitivity of a target layer and nullify sample bias, we require an ample amount of data, which can be a constraint in some scenarios. In light of these challenges, our forthcoming research will prioritize the reduction of data dependency through innovative approaches, including the exploration of techniques such as the utilization of Hessian information from the model parameters. Our primary objective is to augment the robustness and consistency of our methodology, thereby ensuring its continued relevance and popularity in comparison to well-established methods like quantization and pruning. By mitigating data dependency, we aim to make our approach more accessible and adaptable to a wider range of applications.

REFERENCES

[1] S. Zhang, G. Liang, S. Pan, and L. Zheng, "A fast medical image super resolution method based on deep learning network," *IEEE Access*, vol. 7, pp. 12319–12327, 2019.

[2] Y. Xiao, X. Su, Q. Yuan, D. Liu, H. Shen, and L. Zhang, "Satellite video super-resolution via multiscale deformable convolution alignment and temporal grouping projection," *IEEE Trans. Geosci. Remote Sens.*, vol. 60, 2022, Art. no. 5610819.

[3] K. Jiang, Z. Wang, P. Yi, J. Jiang, J. Xiao, and Y. Yao, "Deep distillation recursive network for remote sensing imagery super-resolution," *Remote Sens.*, vol. 10, no. 11, p. 1700, Oct. 2018.

[4] K. Jiang, Z. Wang, P. Yi, T. Lu, J. Jiang, and Z. Xiong, "Dual-path deep fusion network for face image hallucination," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 1, pp. 378–391, Jan. 2022.

[5] J. Kim, J. K. Lee, and K. M. Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 1646–1654.

[6] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee, "Enhanced deep residual networks for single image super-resolution," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jul. 2017, pp. 1132–1140.

[7] K. Jiang, Z. Wang, P. Yi, and J. Jiang, "Hierarchical dense recursive network for image super-resolution," *Pattern Recognit.*, vol. 107, Nov. 2020, Art. no. 107475.

[8] Y. Zhang, Y. Tian, Y. Kong, B. Zhong, and Y. Fu, "Residual dense network for image super-resolution," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2472–2481.

[9] Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong, and Y. Fu, "Image super-resolution using very deep residual channel attention networks," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 286–301.

[10] B. Xia, Y. Hang, Y. Tian, W. Yang, Q. Liao, and J. Zhou, "Efficient non-local contrastive attention for image super-resolution," in *Proc. AAAI Conf. Artif. Intell.*, 2022, vol. 36, no. 3, pp. 2759–2767.

[11] Z. Lu, J. Li, H. Liu, C. Huang, L. Zhang, and T. Zeng, "Transformer for single image super-resolution," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2022, pp. 456–465.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017.

[13] K. Jiang, Z. Wang, P. Yi, G. Wang, T. Lu, and J. Jiang, "Edge-enhanced GAN for remote sensing image superresolution," *IEEE Trans. Geosci. Remote Sens.*, vol. 57, no. 8, pp. 5799–5812, Aug. 2019.

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 27, 2014.

[15] Y. Xiao, Q. Yuan, K. Jiang, J. He, Y. Wang, and L. Zhang, "From degrade to upgrade: Learning a self-supervised degradation guided adaptive network for blind remote sensing image super-resolution," *Inf. Fusion*, vol. 96, pp. 297–311, Aug. 2023.

[16] C. Dong, C. C. Loy, and X. Tang, "Accelerating the super-resolution convolutional neural network," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 391–407.

[17] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2014, pp. 184–199.

[18] W.-S. Lai, J.-B. Huang, N. Ahuja, and M.-H. Yang, "Deep Laplacian pyramid networks for fast and accurate super-resolution," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5835–5843.
[19] N. Ahn, B. Kang, and K.-A. Sohn, "Fast, accurate, and lightweight super-resolution with cascading residual network," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 252–268.
[20] J. Kim, J. K. Lee, and K. M. Lee, "Deeply-recursive convolutional network for image super-resolution," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 1637–1645.
[21] Y. Tai, J. Yang, X. Liu, and C. Xu, "MemNet: A persistent memory network for image restoration," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 4539–4547.
[22] Z. Li, J. Yang, Z. Liu, X. Yang, G. Jeon, and W. Wu, "Feedback network for image super-resolution," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 3862–3871.
[23] Z. Hui, X. Wang, and X. Gao, "Fast and accurate single image super-resolution via information distillation network," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 723–731.
[24] X. Chu, B. Zhang, H. Ma, R. Xu, and Q. Li, "Fast, accurate and lightweight super-resolution with neural architecture search," in *Proc. 25th Int. Conf. Pattern Recognit. (ICPR)*, Jan. 2021, pp. 59–64.
[25] H. Lin and J. Yang, "Light weight IBP deep residual network for image super resolution," *IEEE Access*, vol. 9, pp. 93399–93408, 2021.
[26] X. Chen, J. Chen, X. Han, C. Zhao, D. Zhang, K. Zhu, and Y. Su, "A light-weighted CNN model for wafer structural defect detection," *IEEE Access*, vol. 8, pp. 24006–24018, 2020.
[27] Z. Huang, X. Zhu, M. Ding, and X. Zhang, "Medical image classification using a light-weighted hybrid neural network based on PCANet and DenseNet," *IEEE Access*, vol. 8, pp. 24697–24712, 2020.
[28] I. Mukherjee and S. Tallur, "Light-weight CNN enabled edge-based framework for machine health diagnosis," *IEEE Access*, vol. 9, pp. 84375–84386, 2021.
[29] M. A. Qureshi and A. Munir, "Sparse-PE: A performance-efficient processing engine core for sparse convolutional neural networks," *IEEE Access*, vol. 9, pp. 151458–151475, 2021.
[30] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, and R. Feris, "BlockDrop: Dynamic inference paths in residual networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8817–8826.
[31] S. Teerapittayanon, B. McDanel, and H. T. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," in *Proc. 23rd Int. Conf. Pattern Recognit. (ICPR)*, Dec. 2016, pp. 2464–2469.
[32] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov, "Spatially adaptive computation time for residual networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1790–1799.
[33] T. N. Nguyen, X. T. Nguyen, K. Lee, and H.-J. Lee, "Computation-skipping mask generation for super-resolution networks," in *Proc. Int. Conf. Electron., Inf., Commun. (ICEIC)*, Feb. 2022, pp. 1–3.
[34] L. Wang, X. Dong, Y. Wang, X. Ying, Z. Lin, W. An, and Y. Guo, "Exploring sparsity in image super-resolution for efficient inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 4915–4924.
[35] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, Dec. 2019, pp. 8026–8037.
[36] E. Agustsson and R. Timofte, "NTIRE 2017 challenge on single image super-resolution: Dataset and study," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jul. 2017, pp. 1122–1131.
[37] M. Bevilacqua, A. Roumy, C. Guillemot, and M.-L. Alberi-Morel, "Low-complexity single-image super-resolution based on nonnegative neighbor embedding," in *Proc. BMVC*, 2012, pp. 1–10.
[38] R. Zeyde, M. Elad, and M. Protter, "On single image scale-up using sparse representations," in *Proc. Int. Conf. Curves Surf.*, vol. 6920, 2010, pp. 711–730.
[39] 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," in *Proc. 8th IEEE Int. Conf. Comput. Vis.*, Oct. 2001, pp. 416–423.
[40] J.-B. Huang, A. Singh, and N. Ahuja, "Single image super-resolution from transformed self-exemplars," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 5197–5206.

TUAN NGHIA NGUYEN (Graduate Student Member, IEEE) received the B.S. degree in electronics and computer engineering from the Hanoi University of Science and Technology, Hanoi, Vietnam, in 2017, and the M.S. degree in electrical and computer engineering from Seoul National University, Seoul, South Korea, in 2020, where he is currently pursuing the Ph.D. degree in electrical and computer engineering.

His research interests include computer vision and machine-learning.

XUAN TRUONG NGUYEN (Member, IEEE) received the B.S. degree in electrical engineering from the Hanoi University of Science and Technology, Hanoi, Vietnam, in 2011, and the M.S. and Ph.D. degrees in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2015 and 2019, respectively.

He was a Postdoctoral Fellow with BK21+, the Department of Electrical and Computer Engineering (ECE), and the Inter-University Semiconductor Research Center (ISRC), Seoul National University, from April 2019 to August 2021. Since September 2021, he has been a Research Assistant Professor with the Department of Next-Generation Semiconductor Convergence and Open Sharing Systems (COSS). His research interests include hardware-aware algorithm optimization and system-on-chip (SoC) design for computer vision and multimedia applications.

KYUJOONG LEE (Member, IEEE) received the B.S. degree in electrical engineering from Seoul National University, Seoul, South Korea, in 2002, the M.S. degree in electrical engineering from the University of Southern California, Los Angeles, CA, USA, in 2008, and the Ph.D. degree in electrical and computer engineering from Seoul National University, in 2013.

From 2002 to 2005, he was a Developer with Com2us Corporation, Seoul. From 2013 to 2017, he was with the S.LSI Division, Samsung Electronics Corporation. In 2017, he was appointed as an Assistant Professor with the Department of Electronic Engineering, Sun Moon University, Asan-si, South Korea. In 2022, he was an Associate Professor with the School of AI Convergence, Sungshin Women's University. His major research interests include deep learning algorithms and architectures and image/video compression and processing.

HYUK-JAE LEE (Member, IEEE) received the B.S. and M.S. degrees in electronics engineering from Seoul National University, Seoul, South Korea, in 1987 and 1989, respectively, and the Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 1996.

From 1996 to 1998, he was a Faculty Member with the Department of Computer Science, Louisiana Tech University, Ruston, LA, USA. From 1998 to 2001, he was a Senior Component Design Engineer with the Server and Workstation Chipset Division, Intel Corporation, Hillsboro, OR, USA. In 2001, he joined the School of Electrical Engineering and Computer Science, Seoul National University, where he is currently a Professor. He is the Founder of Mamurian Design Inc., Seoul, which is a fabless SoC design house for multimedia applications. His research interests include computer architecture and SoC designs for multimedia applications.