# Depth-Controllable Very Deep Super-Resolution Network

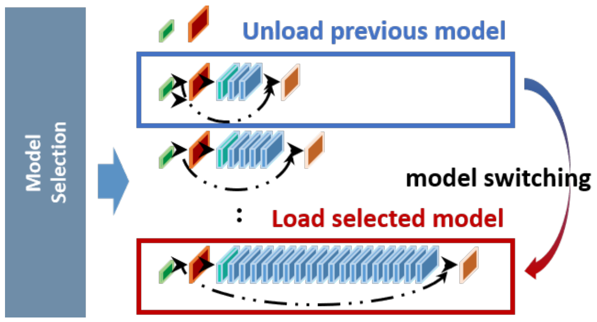
Dohyun Kim
*Chung-Ang University*
Seoul, Korea
dohyun.cau@gmail.com

Joongheon Kim
*Chung-Ang University*
Seoul, Korea
joongheon@cau.ac.kr

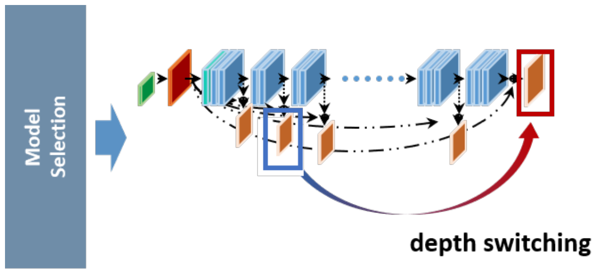
Junseok Kwon
*Chung-Ang University*
Seoul, Korea
jskwon@cau.ac.kr

Tae-Hyung Kim
*KT AI Tech Center*
Seoul, Korea
mrtonnet@gmail.com

***Abstract*—Deep learning techniques not only have surpassed humans in several computer vision tasks, but also have achieved state-of-the-art performance on the super-resolution (SR) task, which enhances the resolution of reconstructed images from the observed low-resolution (LR) images. Very Deep Super-Resolution (VDSR) is a popular architecture with decent performance on the SR task. In general, the deeper the VDSR network, the better the performance. Say, on a computing device with limited computational resources, the SR task must support multiple resolutions or multiple output rates. For this multi-rate SR task, the state-of-the-art architectures require multiple networks with varying depths. However as the number of supported rates increases, so does the number of networks imposing incremental computational burden on the computing device. We propose a depth-controllable network and training principles for the multi-rate SR task. The proposed network is configured as a single network regardless of the number of supported rates. The inverse auxiliary loss and contiguous/progressive skip connections are presented to train the network end-to-end throughout the varying number of layers without biasing the performances of specific depths. With three data sets and three scaling factors, the proposed network is compared to the baseline network VDSR, along with the Super-Resolution Convolutional Neural Network (SRCNN). Our network not only requires a single network at varying rates, but also performs as well as the baseline networks in terms of Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity (SSIM). Our model outperforms the baseline networks when the depth of layers is more than eleven.**



(a) Conventional model switching [10]



(b) Proposed depth switching (DCVDSR)

Fig. 1: Basic concept of DCVDSR. We focus on designing a controllable deep neural network rather than a trade-off optimization algorithm; thus, DCVDSR is free to model selection algorithms.

## I. Introduction

Super-resolution is one of the fundamental and practical problems in the field of computer vision research. Single-image super-resolution (SISR) aims to produce a high-resolution (HR) image from a single low-resolution observation [1]. SISR tries to reconstruct HR images from LR observations that are as close to the real images and as visually pleasant as possible. However, SISR is fundamentally a challenging problem due to the fact that a LR image can correspond to a set of HR images, while most of them are not expected.

Recently, deep-learning-based methods have also achieved great success in super-resolution tasks. Despite the common opinion that deeper neural networks perform better, low-level image processing domains such as super-resolution have been known to suffer stability issues when training them. Dong *et al.* [2] tackled this issue finding that a sequence of convolutional neural networks show great potential for the SR task. Lee *et al.* [3] replaced the 9-1-5 architecture of SRCNN with twenty layers of 3x3 convolution filter and applied global skip connection, based on inspiration from Resnet [11]. As a result, the method outperformed existing approaches by a considerable margin, confirmed the possibility of applying deep neural networks to SR tasks, and greatly inspired subsequent studies [4]–[7]. Although differences exist in their approaches, the aforementioned studies have all contributed significantly to increasing either the depth of neural networks or the stability of training in deeper networks. As the capability of a neural network is proportional to the depth of the network, a trade-off typically exists between a network's depth and its performance and computational cost [8]. Deeper networks perform better, but require more execution time and computational resources. Therefore, selecting a suitable model with proper consideration of these trade-offs has been an important factor in achieving system stability and efficiency. These trade-offs are equally applied to SR tasks, as summarized in [9].

However, determining an optimal model statically can be difficult in some cases, such as when the state of the system or I/O rate changes during operation, or when the system cannot be known in advance (*e.g.* applications for heterogeneous mobile devices). In our previous work [10], we handled this uncertainty by preparing batches of models at various scales and then switching them dynamically to fit the state of the system. Fig. 1(a) shows an example of this model-switching approach with batches of models. However, this approach may incur increased use of memory resources in proportion to the number of models and model-switching overhead, which should be resolved for practical applications. To overcome these limitations, this study aims to achieve the following research objectives: 1) to configure models with different depths as a shared single network in order to remove redundancies on their weights,. and 2) to make the performance of the available depth options in a shared single neural network comparable to each model with a different depth.

Our contribution can be summarized as follows:

- We propose a *depth-controllable very deep super-resolution network* (DCVDSR), which can dynamically select the depth to be used from four layers to twenty layers in a single network, with only minimal additional parameters. Fig. 1(b) shows the depth switching approach of the proposed DCVDSR.
- We propose and analyze a new training strategy using *inverse auxiliary loss* to prevent the performance from favoring a specific depth option while training DCVDSR in an end-to-end manner.
- We evaluate the performance at each depth option of DCVDSR by comparing DCVDSR with baseline models which have corresponding hidden layers. As a result, we achieve performances comparable to the baselines in all depth options of the DCVDSR through the proposed learning method, and even outperform the baselines in depth options deeper than eleven layers.

Note that we focus on designing a controllable deep neural network rather than a trade-off optimization algorithm. The depth-switching step is independent of the model-selection step, making our approach applicable to various algorithms, such as those proposed in [10].

The remainder of this paper is organized as follows: Sec. II describes the proposed approach. Implementation details and experimental results are presented in Sec. III and Sec. IV, respectively. Sec. V concludes this paper.

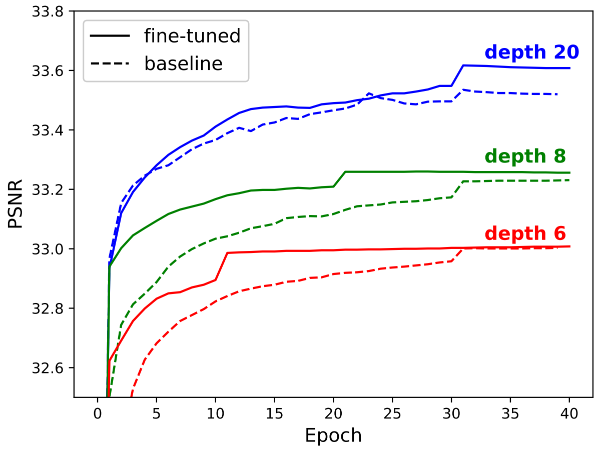


Fig. 2: Motivation of DCVDSR. Experimental results over different depths. Dotted lines denote the results of baseline models, and solid lines denote the results of depth-wise, fine-tuned networks.

## II. Depth Controllable Network

### A. Baseline

We use the VDSR [3] architecture as a baseline because we focus on verifying the applicability and generality of the proposed approaches rather than outperforming existing state-of-the-art performances. In our framework, VDSR can be adopted easily due to its monotonous structure. Moreover, in spite of its simple architecture, VDSR still demonstrates good performance. In contrast to other approaches (especially post-upscaling approaches [6], [7]), VDSR is flexible for different scale factors (from x2 to x4), which enables us to consider additional trade-off factors. Note that VDSR can serve as a connection point to further studies, because it shares theoretical and structural properties with other works [4], [5].

### B. Verification of Motivation

Inspired by Lim *et al.* [12], we fine-tune the weights of the lower layers of deeper networks with the weights pre-learned by other shallower networks, in order to verify the possibility of weight sharing between models with different depths. For example, in a neural network with six layers (*i.e.*, one input, one output, and four hidden layers), the first two layers are copied from two hidden layers which have been pre-trained by the network with four layers. Likewise, in the case of a network with 8 layers, the hidden layers of the 6-layer network are transferred from the shallower network. The remaining layers are initialized arbitrarily. We call this method depth-wise fine-tuning.

If there is no room for sharing weights between the shallower and deeper neural networks, the convergence speed will be equal to or slower than that of the initialized

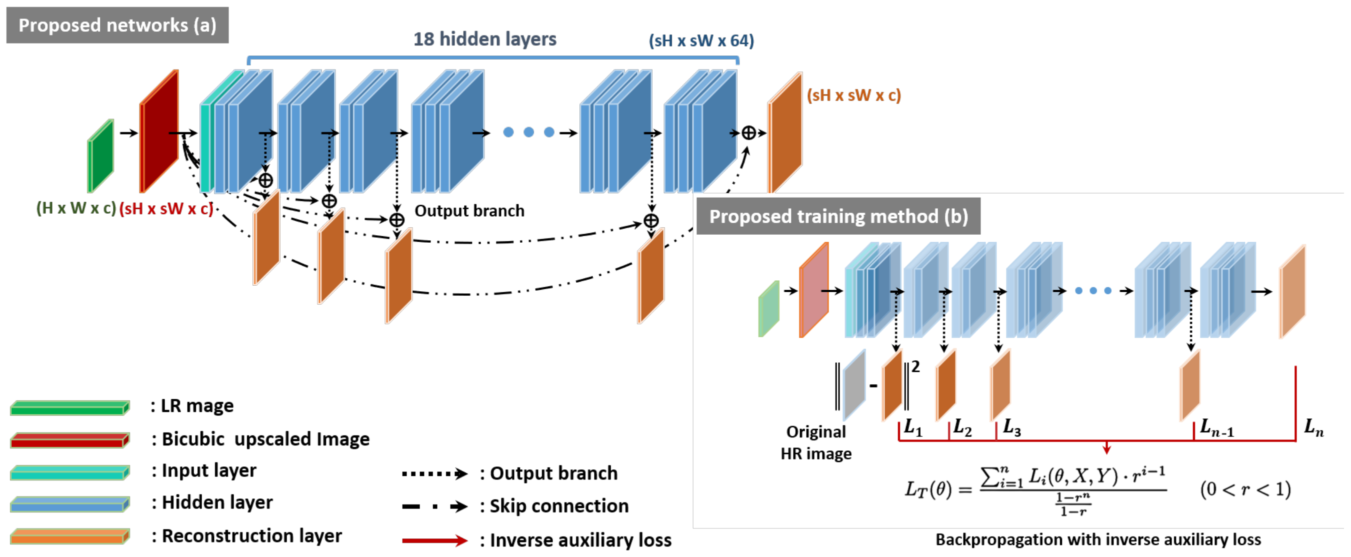


Fig. 3: Architecture of DCVDSR. (a) DCVDSR contains several output branches which consist of convolutional layers that are the same as the output layer, and each output branch is connected with the interpolated low-resolution image, (b) our training strategy using inverse auxiliary loss..

neural network. However, we observe that in all cases, as shown in Fig. 2, the network trained with depth-wise fine-tuning converges faster, thereby substantiating our assumption. Interestingly, both the convergence speed and the restoration performance improve. This improvement is particularly noticeable when using deeper neural networks, thereby suggesting the potential for depth-wise fine-tuning to improve the stability of training.

### C. Output Branch and Multi-Scale Loss

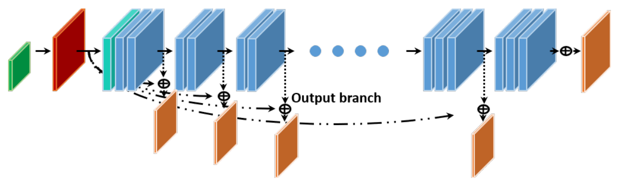
Based on the above observations, we propose adding output branches in the middle of the hidden layers of VDSR. Fig. 3(a) shows the overall architecture. These output branches consist of the $3 \times 3 \times 64 \times c$ convolution filters, just like the output layer of VDSR, where c denotes the dimension of the target image. The output branches have two-fold responsibilities. First, the output branch transforms an intermediate feature map into a shape of the target image. Second, each output branch carries a different part of the weights, which cannot be shared across the entire network. Like conventional trade-off problems, the ratio of unbound weights increases as the depth of the output branch increases. Thus, the performance of each branch can be improved. However, this also causes the number of parameters to increase. To solve this problem, we allow each output branch to have only one layer.

As previously mentioned, because the shape of the outputs obtained from each branch is the same as the target image, we calculate the losses from all branches and exploit them for training. This is conceptually similar to the auxiliary classifier of the Inception network [13], and the multi-scale loss proposed in [14], [15]. The auxiliary classifier [13] is used to alleviate a gradient vanishing problem in lower layers by multiplying the loss of each auxiliary branch by the decay rate $0.3^n$, where $n$ denotes the order of the branch from the output layer. Unlike the Inception network and the multi-scale loss in [14], [15], each output branch in the proposed DCVDSR has the explicit role of outputting the image in each branch, so it seems more reasonable that the losses should be averaged rather than decayed. However, the averaged loss still contains some problems. To handle these problems, we propose a novel training strategy that uses the inverse auxiliary loss, as discussed in Sec. II-E.

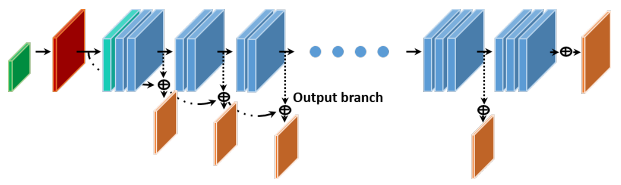
### D. Contiguous and Progressive Skip Connection

VDSR uses a global skip connection to add the up-scaled input image to the result of the output layer using bicubic interpolation, and this improves the stability of training. However, because other output branches exist in DCVDSR, distortion may occur among the branches during training, if skip connection is applied only to the last output layer. In the case of training with the loss at the last output layer, for example, the weights are trained to reflect only the difference from the input image due to the skip connection. However, in other output branches, training is performed to produce output images from scratch, so the extent to which weights can be updated may be relatively larger than that of the previous layer. Therefore, in order to maintain consistency and to share the weights properly, the skip connection should be applied to all of the branches. Fig. 4 shows two ways of the skip connection methods.

Fig. 4(a) depicts the case in which global skip connection is applied to all of the output branches, as in the output layer. Fig. 4(b) depicts the case in which the result of the previous branch is progressively added to the result of the next branch. We call these two approaches as *contiguous*



(a) Contiguous skip connection



(b) Progressive skip connection

Fig. 4: (a) contiguous skip connection, in which the global skip connection is applied to all of the output branches, as in the output layer; (b) progressive skip connection, in which the result of the previous branch is progressively added to the result of the next branch.

and *progressive skip connection*, respectively. Eqs.(1) and (2) denote the equations of the contiguous and progressive skip connections at the second output branch, respectively.

$$G_{cont}(X) = O_2(H_2(H_1(X))) + X, \tag{1}$$

$$G_{prog}(X) = O_2(H_2(H_1(X))) + O_1(H_1(X)) + X, \tag{2}$$

where $X$, $G_{cont}$, $G_{prog}$, $O_n$, and $H_n$ denote the upscaled input image, result of contiguous skip connection, result of progressive skip connection, output of $n$-th output branch, and output of the $n$-th hidden layer, respectively.

If the neural network is well trained, the result of $O_1(H_1(X)) + X$ in (2) will be closer to the ground-truth image than $X$. Therefore, $O_2(H_2(H_1(X)))$ in (2) has an advantage by training a smaller residual than (1). Nonetheless, the progressive skip connection performs worse than the contiguous skip connection, as shown in Table I. This is presumably because the progressive skip connection is dependent on the previous branches, so the training at the shallower branches can be interrupted by the deeper branches. We reserve further analysis on this issue for future work and use the contiguous skip connection in this study.

### *E. Inverse Auxiliary Loss*

The proposed model trained using the averaged loss demonstrates performance comparable to that of the baselines in the middle branches (depth 8 to 11) and even outperforms the baselines in the deeper branches (depth 14 to 20). However, in the shallow branches (depth 4 to 6), considerable performance degradation occurs compared to the baselines (over 0.2dB). As in the Inception network, the loss of the shallow branches can help to improve the stability of training at the deeper branches, while the convergence of the shallower branches can be disturbed by the deeper branches. Therefore, in order to achieve performance comparable to the baseline over the entire interval, we need an advanced approach of training the shallower branches explicitly.

For this purpose, it is worth considering stacked transfer learning, which trains a shallow layer and then sequentially trains other hidden layers and branches, while freezing the pretrained shallower one. This approach is advantageous because it makes the training procedure at shallower depths more explicit, and the training at the deeper branches proceeds after the shallower branches have completely converged. However, the sharing of weights with the deeper layers cannot be taken into account in this approach. Thus, it is difficult to expect performance improvement as the network becomes deeper (see stack transfer (hard) in Table II, in which there is no performance improvement from depth 8).

It is effective to stop the pre-training at shallower branches earlier, before the network completely converges. Stopping early causes the deeper branches to share the weights, while preventing the weights from fitting too hard to the shallower branches The *stacked transfer(soft)* of Table II shows promising results, with the performances slightly degrading at the shallower branches and increasing at the deeper branches. Thus, we propose an *inverse averaged loss* to take advantage of both the averaged loss and the stacked transfer learning. The proposed method is shown in Fig. 3(b).

Eq.(3) is the inverse averaged loss.

$$L_T(\theta) = \frac{\sum_{i=1}^{n} L_i(\theta, X, Y) \cdot r^{i-1}}{\frac{1-r^n}{1-r}} \qquad (0 < r < 1). \tag{3}$$

In contrast to the auxiliary classifier of the Inception network, the proposed loss is multiplied by the decay coefficient r as it moves into the deeper branches. The summed loss is divided by the sum of the coefficients; then, the weights are updated using the inverse averaged loss. Unlike the averaged loss, the inverse averaged loss for the shallower branches can be emphasized explicitly because the effect of each branch exponentially decreases as the layer becomes deeper. Moreover, the neural network converges more easily over the shared weights because the weights of the deeper branches are warmed up in advance. As shown in Table II, we not only achieve performance at the shallow branches comparable to the baselines, but also outperform the baselines at the deep branches. Note that (3) is equivalent to the averaged loss if the decay coefficient $r$ is 1, and to the stacked transfer learning, if $r$ is zero. We also conducted an experiment on the effect of the decay coefficient on training, and the results are analyzed in Sec. IV.

## III. Implementation Details

For fair comparison, we maintain the structure of the VDSR as fully as possible. The upscaled image produced using bicubic interpolation is used as an input. The input, hidden, and output layers consist of $w \times h \times c \times 64$, $w \times h \times$

TABLE I: results of contiguous and progressive skip connection. We compare our methods against two baselines, the original VDSR and the re-implemented version of VDSR using TensorFlow. To compare the shallowest networks, we use SRCNN, which is based on VDSR.

| Model | Depth | | | | | | |
|---|---|---|---|---|---|---|---|
| | 0 (Bicubic) PSNR/SSIM | 4 (SRCNN) PSNR/SSIM | 6 PSNR/SSIM | 8 PSNR/SSIM | 11 PSNR/SSIM | 14 PSNR/SSIM | 20 (VDSR) PSNR/SSIM |
| Baseline (original) | 30.4 / 0.8682 | 32.7 / 0.909 | - | - | - | - | 33.6 / 0.921 |
| Baseline (implement) | 30.4 / 0.8682 | 32.56/0.91 | 33.01/0.916 | 33.229/0.918 | 33.379/0.92 | 33.435/0.92 | 33.523/0.921 |
| Averaged loss (Cont) | 30.4 / 0.8682 | 32.19/0.9058 | 32.79/0.9128 | 33.11/0.9164 | 33.3/0.9187 | 33.48/0.9208 | 33.59/0.9222 |
| Averaged loss (Prog) | 30.4 / 0.8682 | 32.182/0.905 | 32.716/0.912 | 33.06/0.916 | 33.231/0.917 | 33.356/0.919 | 33.446/0.92 |

TABLE II: Quantitative results of stacked transfer learning approaches. Stacked transfer (soft) denotes the results obtained by transferring the early stopped pre-trained model (20 epochs) to lower layers of a deeper network. Stacked transfer (hard) denotes the results obtained by transferring the hardly converged pre-trained model (50 epochs). When the PSNR score is equal to or better than the baseline (our implementation), the results are presented in blue. When the PSNR score is less than 0.3dB compared to the baseline (original paper), the results are presented in red.

| Model | Depth | | | | | | |
|---|---|---|---|---|---|---|---|
| | 0 (Bicubic) PSNR/SSIM | 4 (SRCNN) PSNR/SSIM | 6 PSNR/SSIM | 8 PSNR/SSIM | 11 PSNR/SSIM | 14 PSNR/SSIM | 20 (VDSR) PSNR/SSIM |
| Baseline (original) | 30.4 / 0.8682 | 32.7 / 0.909 | - | - | - | - | 33.6 / 0.921 |
| Baseline (implement) | 30.4 / 0.8682 | 32.56/0.91 | 33.01/0.916 | 33.229/0.918 | 33.379/0.92 | 33.435/0.92 | 33.523/0.921 |
| Averaged loss | 30.4 / 0.8682 | 32.19/0.9058 | 32.79/0.9128 | 33.11/0.9164 | 33.3/0.9187 | 33.48/0.9208 | 33.59/0.9222 |
| Stacked transfer (soft) | 30.4 / 0.8682 | 32.47/0.91 | 32.86/0.9140 | 32.952/0.9148 | 32.959/0.9149 | 32.96/0.9149 | 32.96/0.914 |
| Stacked transfer (hard) | 30.4 / 0.8682 | 32.56/0.91 | 32.68/0.9121 | 32.72/0.9125 | 32.724/0.9125 | 32.73/0.9126 | 32.73/0.9126 |
| Inverse aux loss (0.5) | 30.4 / 0.8682 | 32.43/0.9093 | 32.97/0.915 | 33.18/0.9172 | 33.38/0.9195 | 33.49/0.9207 | 33.57/0.9214 |

$64 \times 64$, and $w \times h \times 64 \times c$ weights, respectively, where $w$, $h$ and $c$ denote width, height, and the number of channels of target image, respectively. Like the output layer, the output branches consist of a single convolution layer with $w \times h \times 64 \times c$ weights. In this study, we added output branches to 4, 6, 8, 11, 14, and 17th hidden layer, but the position and interval are freely configurable. The only difference with VDSR is the Adam optimizer [16]. We used its parameters same as [16]. The learning rate begins at 0.0001 and decreased 10 times by 30th epoch. Training is performed during the 50th epoch with batch sizes of 128. Although the L1 loss is typically more effective than the L2 loss, we use the L2 loss, as in VDSR, for fair comparison.

All deep neural networks have been re-implemented on the TensorFlow framework, and some performance degradation has occurred, but without affecting the experiments.

## IV. Experiments

In this section, we quantitatively and qualitatively evaluate and analyze the proposed DCVDSR. For quantitative evaluation, we use PSNR and SSIM metrics, which have been adopted widely in the image restoration field [17], [18]. For both metrics, a higher score means better visual quality.

### A. Experimental Settings

For a fair comparison, we follow the same settings used in VDSR. For training, we extract patches with a stride size of 20 in $40 \times 40$ size from 291 images and apply the data augmentation with scaled (by $\times 2$, $\times 3$ and $\times 4$), rotated (by degree of $90, 180, 270$) and flipped images. As a benchmark, we use Set5 [19], Set14 [20], and BSD100 [21], which contain 5 and 14 color images in various sizes (from $228 \times 228$ to $512 \times 512$), and 100 color images of $480 \times 320$ or $320 \times 480$ sizes, respectively. Note that the inference step of DCVDSR is independent of the size of the input images, just as in VDSR. All results shown in the previous sections are evaluated using the Set5 dataset. All of the data were evaluated in a single batch and the average of the scores was used for each measurements.

### B. Quantitative Evaluation

The quantitative evaluation is performed in three steps: 1) all results are compared to the baselines; 2) The results are compared using the averaged loss and the inverse auxiliary loss in the proposed model structure; and 3) the results of varying the decay coefficients of the auxiliary loss from 0.3 to 0.7 are compared;

Table III shows the experimental results of the proposed method compared to the baselines with various settings. Each column represents the PSRN and SSIM scores using difference scales, test data sets, and depths. For the baseline (implement), the unshared VDSR-based model is trained for each depth. The *averaged loss* and the *inverse auxiliary loss (0.5)* denote the results of the corresponding output branch for each depth in the proposed single model trained using the averaged loss and the inverse auxiliary loss of the decaying rate 0.5, respectively. In particular, when the PSNR outperforms the baseline (implement), the

TABLE III: Ablation study on the same model without training for different factors (scales, depths, and datasets). When our method outperforms the baseline (our implementation), the results are presented in blue, and when the PSNR score is less than 0.3dB compared to the baseline, the results are presented in red. The processing time is evaluated using *"Monarch.bmp"* in Set5, for which the resolution is 256 × 256

| Model | | Depth | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 0 (Bicubic) PSNR/SSIM | 4 (SRCNN) PSNR/SSIM | 6 PSNR/SSIM | 8 PSNR/SSIM | 11 PSNR/SSIM | 14 PSNR/SSIM | 20 (VDSR) PSNR/SSIM |
| Set5 | | | | | | | | |
| Baseline (original) | x 2 | 33.66/0.9299 | 36.66/0.9542 | - | - | - | - | 37.53/0.9587 |
| | x 3 | 30.40/0.8682 | 32.7/0.909 | - | - | - | - | 33.6/0.921 |
| | x 4 | 28.42/0.8104 | 30.48/0.8628 | - | - | - | - | 31.35/0.8838 |
| Baseline (implemented) | x 2 | 33.66/0.9299 | 36.56/0.9551 | 37.05/0.9577 | 37.22/0.9584 | 37.33/0.9589 | 37.36/0.9591 | 37.4/0.9592 |
| | x 3 | 30.40/0.8682 | 32.56/0.91 | 33.01/0.916 | 33.229/0.918 | 33.379/0.92 | 33.435/0.92 | 33.523/0.921 |
| | x 4 | 28.42/0.8104 | 30.29/0.8649 | 30.64/0.8736 | 30.84/0.8774 | 30.97/0.8795 | 31.07/0.8816 | 31.15/0.8837 |
| Averaged loss | x 2 | 33.66/0.9299 | 35.59/0.931 | 36.82/0.9501 | 37.13/0.9563 | 37.27/0.957 | 37.39/0.9582 | 37.45/0.9595 |
| | x 3 | 30.40/0.8682 | 32.19/0.9058 | 32.79/0.9128 | 33.11/0.9164 | 33.3/0.9187 | 33.48/0.9208 | 33.59/0.9222 |
| | x 4 | 28.42/0.8104 | 29.81/0.8523 | 30.45/0.869 | 30.7/0.8745 | 30.89/0.8784 | 31.12/0.882 | 31.26/0.8845 |
| Inverse auxiliary loss (0.5) | x 2 | 33.66/0.9299 | 36.29/0.9537 | 36.98/0.9572 | 37.18/0.9582 | 37.31/0.9587 | 37.38.0.959 | 37.41.0.9597 |
| | x 3 | 30.40/0.8682 | 32.43/0.9093 | 32.97/0.915 | 33.18/0.9172 | 33.38/0.9201 | 33.49/0.9207 | 33.57/0.9214 |
| | x 4 | 28.42/0.8104 | 30.13/0.8612 | 30.58/0.8722 | 30.79/0.8726 | 31.01/0.8761 | 31.15/0.8821 | 31.23/0.8833 |
| Set14 | | | | | | | | |
| Baseline (original) | x 2 | 30.24/0.8688 | 32.42/0.9063 | - | - | - | - | 33.03/0.9124 |
| | x 3 | 27.55/0.7742 | 29.28/0.8209 | - | - | - | - | 29.77/0.8314 |
| | x 4 | 26.00/0.7027 | 27.49/0.7503 | - | - | - | - | 28.01/0.7674 |
| Baseline (implemented) | x 2 | 30.24/0.8688 | 32.46/0.9064 | 32.8/0.9099 | 32.9/0.9106 | 32.99/0.9112 | 33.01/0.9115 | 33.03/0.9117 |
| | x 3 | 27.55/0.7742 | 29.17/0.8195 | 29.47/0.8251 | 29.56/0.8273 | 29.66/0.8285 | 29.71/0.8296 | 29.71/0.8301 |
| | x 4 | 26.00/0.7027 | 27.34/0.7480 | 27.58/0.7567 | 27.7/0.7593 | 27.76/0.7614 | 27.8/0.7632 | 27.84/0.7639 |
| Averaged loss | x 2 | 30.24/0.8688 | 32.01/0.9022 | 32.65/0.9082 | 32.85/0.9101 | 32.94/0.911 | 33.03/0.9117 | 33.08/0.9123 |
| | x 3 | 27.55/0.7742 | 28.95/0.8147 | 29.34/0.8222 | 29.52/0.8259 | 29.63/0.8279 | 29.72/0.8299 | 29.76/0.8309 |
| | x 4 | 26.00/0.7027 | 26.99/0.7356 | 27.46/0.7525 | 27.63/0.7573 | 27.74/0.7605 | 27.84/0.7634 | 27.9/0.7656 |
| Inverse auxiliary loss (0.5) | x 2 | 30.24/0.8688 | 32.31/0.9053 | 32.74/0.9093 | 32.88/0.9104 | 32.98/0.9112 | 33.02/0.9116 | 33.04/0.9119 |
| | x 3 | 27.55/0.7742 | 29.12/0.8191 | 29.43/0.8241 | 29.57/0.8266 | 29.67/0.8286 | 29.72/0.8295 | 29.74/0.9298 |
| | x 4 | 26.00/0.7027 | 27.22/0.7444 | 27.51/0.7549 | 27.66/0.7587 | 27.79/0.762 | 27.85/0.7635 | 27.88/0.7644 |
| BSD100 | | | | | | | | |
| Baseline (original) | x 2 | 29.56/0.8431 | 31.36/0.8879 | - | - | - | - | 31.90/0.8960 |
| | x 3 | 27.21/0.7385 | 28.41/0.7863 | - | - | - | - | 28.82/0.7976 |
| | x 4 | 25.96/0.6675 | 26.90/0.7107 | - | - | - | - | 27.89/0.7251 |
| Baseline (implemented) | x 2 | 29.56/0.8431 | 31.30/0.889 | 31.56/0.893 | 31.65/0.8941 | 31.71/0.895 | 31.76/0.8954 | 31.76/0.8958 |
| | x 3 | 27.21/0.7385 | 28.26/0.7853 | 28.466/0.791 | 28.56/0.7932 | 28.61/0.7947 | 28.65/0.7959 | 28.66/0.7965 |
| | x 4 | 25.96/0.6675 | 26.75/0.7082 | 26.92/0.7166 | 27/0.7188 | 27.05/0.7207 | 27.08/0.7221 | 27.1/0.7231 |
| Averaged loss | x 2 | 29.56/0.8431 | 31/0.8849 | 31.42/0.8908 | 31.59/0.8933 | 31.68/0.8946 | 31.75/0.8956 | 31.8/0.8963 |
| | x 3 | 27.21/0.7385 | 28.1/0.7804 | 28.37/0.7879 | 28.49/0.7916 | 28.58/0.7938 | 28.65/0.7961 | 28.7/0.7974 |
| | x 4 | 25.96/0.6675 | 26.51/0.6966 | 26.83/0.7128 | 26.94/0.7171 | 27.02/0.7198 | 27.1.0.7225 | 27.14/0.7244 |
| inverse auxiliary loss (0.5) | x 2 | 29.56/0.8431 | 31.2/0.8879 | 31.5/0.8922 | 31.61/0.8937 | 31.69/0.8947 | 31.74/0.8952 | 31.75/0.8955 |
| | x 3 | 27.21/0.7385 | 28.22/0.7851 | 28.44/0.7898 | 28.54/0.7925 | 28.62/0.7947 | 28.66/0.7958 | 28.67/0.7964 |
| | x 4 | 25.96/0.6675 | 26.67/0.7051 | 26.89/0.7151 | 26.97/0.7179 | 27.05/0.7209 | 27.09/0.7222 | 27.1/0.7231 |
| *"Monarch.bmp"* (256 × 256) in Set5 | | | | | | | | |
| Processing time (sec) | CPU | 0.001 | 0.321 | 0.5468 | 0.7725 | 0.994 | 1.317 | 1.96 |
| | GPU | 0.001 | 0.01 | 0.012 | 0.0152 | 0.0189 | 0.0224 | 0.0305 |

results are presented in blue, and when the PSNR score is less than 0.3dB compared to the baseline, the results are presented in red. All approaches are evaluated using the same model without training for different factors, i.e., scales, depths, and datasets.

For the averaged loss, our method performs similar to the baseline (implement), with a difference of less than 0.1db at depths from 8 to 11, and it even outperforms the baseline at a depth of 14. However, it performs worse than the baseline in the shallow section, below a depth of 6 up to 1dB. These results imply that the averaged losses can serve as auxiliaries to stabilize the training of the weights of the deep layers, while weights of the shallow layers deteriorate during the process of converging the deeper layers. Based on the aforementioned results, we design the inverse auxiliary loss to compensate for the shortcomings of the averaged loss and stacked transfer learning methods, and to exploit their merits by multiplying the decay coefficient to the deeper layers in order to explicitly train the shallow layers, while warming up the deeper layers for the subsequent training process. Doing so allows us to achieve an average improvement of 0.3dB in the shallow layer compared to the case trained by only the averaged loss. Although the performance is degraded to some extent at the deep layer, it still outperforms the baselines. Therefore, the inverse auxiliary loss is more appropriate for the

TABLE IV: Ablation study by increasing the values of auxiliary coefficients from 0.3 to 0.7. The best result of each coefficient is presented in blue, and the worst result is in red. When outperforming the baselines (implement), those results are in bold. We also conduct two additional experiments on fine-tuned models using the averaged loss for pre-training and the inverse auxiliary loss for fine-tuning, and on the models using deeper output branches.

| Model | Depth | | | | | | |
|---|---|---|---|---|---|---|---|
| | 0 (Bicubic) | 4 (SRCNN) | 6 | 8 | 11 | 14 | 20 (VDSR) |
| | PSNR/SSIM | PSNR/SSIM | PSNR/SSIM | PSNR/SSIM | PSNR/SSIM | PSNR/SSIM | PSNR/SSIM |
| Baseline (original) | 30.4 / 0.8682 | 32.7 / 0.909 | - | - | - | - | 33.6 / 0.921 |
| Baseline (implement) | 30.4 / 0.8682 | 32.56/0.91 | 33.01/0.916 | 33.229/0.918 | 33.379/0.92 | 33.435/0.92 | 33.523/0.921 |
| Averaged loss | 30.4 / 0.8682 | 32.19/0.9058 | 32.79/0.9128 | 33.11/0.9164 | 33.3/0.9187 | 33.48/0.9208 | 33.59/0.9222 |
| auxiliary coef 0.7 | 30.4 / 0.8682 | 32.34/0.9079 | 32.9/0.9141 | 33.16/0.917 | 33.37/0.919 | **33.51/0.921** | **33.58/0.922** |
| auxiliary coef 0.5 | 30.4 / 0.8682 | 32.43/0.9093 | **32.97/0.915** | 33.18/0.9172 | 33.38/0.9195 | 33.49/0.9207 | 33.57/0.9214 |
| auxiliary coef 0.3 | 30.4 / 0.8682 | 32.45/0.9092 | 32.907/0.9148 | 33.25/0.9171 | 33.33/0.9193 | 33.44/0.9203 | 33.47/0.9206 |
| Staked transfer (soft) | 30.4 / 0.8682 | 32.47/0.91 | 32.86/0.9140 | 32.952/0.9148 | 32.959/0.9149 | 32.96/0.9149 | 32.965/0.914 |

goals of this paper. These results remain consistent for all other data sets and scales; therefore, the proposed method is appropriate as a generic method to achieve our goals and is scalable for a range of scales, just like VDSR. We also evaluated the processing time at each depth with *"Monarch.bmp"* in the Set5 dataset, of which resolution is $256 \times 256$. As shown in bottom of Table III, the shallowest neural network required approximately 6 times less processing time than the deepest neural network when using a CPU, and approximately 3 times less when using a GPU.

Table IV shows the effect of the decay coefficient of inverse averaged loss. As the decay coefficient of the inverse averaged loss approaches 1, it becomes equal to the averaged loss; when it is close to 0, it becomes equal to the stacked transfer learning. Table IV shows the result of a comparative experiment in which we increase the value from 0.3 to 0.7 in order to analyze the effect of this coefficient. To facilitate analysis, the best performance of each coefficient is presented in blue, the worst performance is in red, and when outperforming the baseline (implement), the results are in bold. As shown in Table IV, the performance at shallower layers improves, and the performance at deeper layers worsens, as the value of the coefficient increases. These results indicate that the inverse auxiliary loss plays a significant role, thereby allowing us to adjust the performance in the deep and shallow layers. We use 0.5 as a coefficient, which is empirically balanced and plausible in layers at all depths.

### C. Qualitative Evaluation

We conduct a qualitative evaluation of the proposed DCVDSR with the following considerations: 1) whether the proposed model has a visually plausible ability to up-scale low-resolution images, and 2) whether the proposed model possesses trade-off factors, *i.e.*, its visual quality is improved progressively as the depth of layers becomes deeper.

We perform up-scaling using a different branch of each depth (0, 4, 8, 20) for patches of three images, namely, baboon, monarch butterfly, and PPT3, which are widely used for evaluating super-resolution tasks. Note that it is equivalent to the bicubic interpolation method, especially at a depth of 0. We firstly obtain low-resolution images down-scaled by bicubic interpolation methods with a scaling factor $\times 3$ from the original images, and then we up-scale them using DCVDSR with the corresponding scaling factor. As shown in Fig.5, the output branch at the shallowest depth performs significantly better than the bicubic interpolation, and the quality gradually improves as the layer becomes deeper. The improvement is distinct from that in the shallow layer; however, as the layer becomes deeper, the difference can be compromised according to the purpose. Therefore, for practical purposes, it may be reasonable to use only up to 11 layers when computational resources are insufficient.

## V. Conclusions

The proposed *depth-controllable very deep super-resolution network* has a convenient property that can scale the processing speed and image quality, given a low-resolution input image, by controlling the depth or the number of hidden layers of a single convolutional neural network. The output images from the intermediate hidden layers are enhanced as the number of intermediate layers increases. The network at a specified depth (e.g., twenty layers) is successfully trained while also being able to output up-sampled images at the intermediate stages. For example, the presented intermediate stages are layers 4, 6, 8, 11, and 14. The proposed network at those depths is compared to the baseline networks, mainly to VDSR and SRCNN at a depth of 4, at three scaling factors (x2, X3, x4). We also propose a novel training strategy that uses the inverse auxiliary loss and contiguous/progressive skip connections. The proposed training principles help in training the network end-to-end without biasing the performance at a specific layer.

In conclusion, the proposed network was designed so that shallower parts of the network work independently of deeper parts within a single network. We believe this modular characteristic is especially suitable for practical applications with limited computational resources.

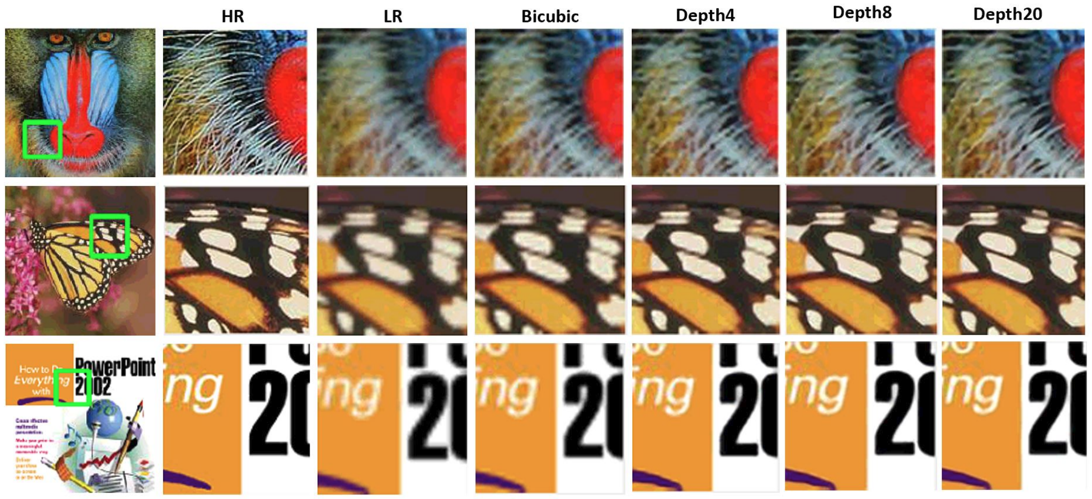


Fig. 5: Results of qualitative evaluation. Evaluation was performed at a different branch of each depth (0, 4, 8, 20) for patches of three images, namely, baboon, monarch butterfly, and PPT3. Note that it is exactly equivalent to the bicubic interpolation, especially at a depth of 0. Low-resolution images are obtained from original images by down-sampling using the bicubic interpolation method with a scaling factor ×3, and then up-sampling was performed using DCVDSR with a corresponding scaling factor.

## Acknowledgment

This research was supported by the National Research Foundation of Korea (2016R1C1B1015406, 2017R1A4A1015675); and also by Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No.2018-0-00170, Virtual Presence in Moving Objects through 5G). J. Kim is the corresponding author of this paper.

## References

[1] K. Nasrollahi and T. B. Moeslund, "Super-resolution: a comprehensive survey," Machine Vision and Applications, vol. 25, pp.1423–1468, August 2014.
[2] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," IEEE Trans. Pattern Anal. Mach. Intell, vol. 38, pp.295–307, 2016.
[3] J. Kim, J. Lee, and K. Lee, "Accurate Image Super-Resolution Using Very Deep Convolutional Networks," IEEE Conf. Computer Vision and Pattern Recognition, pp.1646-1654, 2016
[4] J. Kim, J. Lee, and K. Lee, "Deeply-recursive convolutional network for image super-resolution," IEEE Conf. Computer Vision and Pattern Recognition (CVPR), pp.1637–1645, 2016.
[5] Y. Tai, J. Yang and X. Liu, "Image super-resolution via deep recursive residual network," IEEE Conf. Computer Vision and Pattern Recognition (CVPR), pp.2790–2798, 2017.
[6] T. Tong, G. Li, X. Liu, and Q. Gao, "Image super-resolution using dense skip connections," IEEE Int'l Conf. Computer Vision (ICCV), pp.4809–4817, 2017.
[7] Y. Zhang, Y. Tian, Y. Kong, B. Zhong, and Y. Fu, "Residual dense network for image super-resolution," IEEE Conf. Computer Vision and Pattern Recognition (CVPR), June 2018.
[8] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S.Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," IEEE Conf. Computer Vision and Pattern Recognition (CVPR), pp.3296–3297, 2017.
[9] R. Timofte, E. Agustsson, L. V. Gool, M. H. Yang, L. Zhang, et al., "Ntire 2017 challenge on single image super-resolution: Methods and results," IEEE Conf. Computer Vision and Pattern Recognition (CVPR) Worshops, pp.1110–1121, 2017.
[10] D. Kim, J. Kwon, and J. Kim, "Low-complexity online model selection with lyapunov control for reward maximization in stabilized real-time deep learning platforms," IEEE Int'l Conf. Systems, Man, and Cybernetics (SMC), 2018.
[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," IEEE Conf. Computer Vision and Pattern Recognition (CVPR), pp.770–778, 2016.
[12] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee, "Enhanced deep residual networks for single image super-resolution," IEEE Conf. Computer Vision and Pattern Recognition (CVPR), pp.1132–1140, 2017.
[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," IEEE Conf. Computer Vision and Pattern Recognition (CVPR), pp.2818–2826, 2016.
[14] L. Cavigelli, P. Hager, and L. Benini, "Cas-cnn: A deep convolutional neural network for image compression artifact suppression," IEEE Int'l Joint Conf. Neural Networks (IJCNN), pp. 752–759, 2017.
[15] Y. Hu, X. Gao, J. Li, Y. Huang, and H. Wang, "Single image super-resolution via cascaded multi-scale cross network," Computing Research Repository (CoRR), vol.abs/1802.08808, 2018.
[16] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization," Dec. 2014. [Online]. Available: arXiv preprint arXiv:1412.6980v9.
[17] T. A. Soomro and J. Gao, "Neural network based denoised methods for retinal fundus images and MRI brain images," IEEE Int'l Joint Conf. Neural Networks (IJCNN), pp.1151–1157, 2016.
[18] J. Li, S. You, and A. Robles-Kelly, "A Frequency Domain Neural Network for Fast Image Super-resolution," IEEE Int'l Joint Conf. Neural Networks (IJCNN), pp.1–8, 2018.
[19] M. Bevilacqua, A. Roumy, C.Guillemot, and M. Albei-Morel, "Low-complexity single-image super-resolution based on non-negative neighbor embedding," British Machine Vision Conference (BMVC), pp.1–10, 2012.
[20] R. Zeyde, M. Elad, and M. Protter, "On single image scale-up using sparse-representations," Curves and Surfaces - 7th Int'l Conf., vol.6920, pp.711-730, 2012.
[21] R. Timofte, V. D. Smet, and L. V. Gool, "A+: Adjusted anchored neighborhood regression for fast super-resolution," Asian Conf. Computer Vision (ACCV), pp.111–126, 2014.