APPLIED MACHINE LEARNING SYSTEMS II (ELEC0135) 21/22 REPORT

SN: 20057524

ABSTRACT

Research about deep learning-based single-image super-resolution (SISR) methods has boosted the performance of SISR tasks to a large extent. This article follows Ledig et al. [1] to build an SR-Generative Adversarial Network (SR-GAN) for generating realistic SR images. As a comparison, we built an SR-CNN model with residual blocks (SRResnet) that has the same network structure as the SRGAN generator but uses a different loss function. The results mainly show that SRGAN generates more realistic SR outputs than SR-Resnet. And both models' performances degrade with the upscale factor growing from 2 to 4. [1]

***Index Terms*—** SRGAN, SRResnet, SISR, DIV2K

1. INTRODUCTION

Super-resolution (SR) is the process of restoring high-resolution (HR) images from one or more low-resolution (LR) images of the same content. It is of significant meaning to a wide range of real-world applications, such as old photo restoration, object detection [2], medical care [3], satellite imaging [4], etc. SR can be classified into single-image super-resolution (SISR) and multi-image super-resolution (MISR), in which SISR is generally more efficient than MISR. The fast development of Deep Learning (DL) techniques in recent years has brought prosperous contributions to improving the performance of SISR tasks, such as SRCNN [5] and SRGAN [1]. Due to the SRGAN model's ability to generate more realistic SR images than classical CNNs, we follow Ledig et al. [1] to explore the principles of SRGAN, implement it with Python, and report the related results using PSNR and SSIM metrics with DIV2K dataset. As a comparison, we also explore and implement a PSNR-oriented SRResnet model, which has the same network structure as the generator in SRGAN but has different optimization loss.

This article is organized as the following contents for the next parts: (1) Literature survey of general SR network frameworks, advanced network design, and popular optimization loss functions. (2) Description of the SRResnet and SRGAN. (3) Implementation of data pre-processing, SRResnet, and SRGAN. This section also presents the training and validating results. (4) Testing results, discussion of model performance, and future improvements. (5) Conclusion. (6) Reference.

[1]The code is provided as a GitHub project: AMLS_II_assignment21_22

2. LITERATURE SURVEY

Of all the related SISR algorithms, there are three mainstreams: **interpolation-based methods**, **reconstruction-based methods**, and **example-based methods**. **Interpolation-based methods** are speedy and straightforward but generally cannot reach high accuracies, such as FFT multichannel interpolation [6] and bicubic interpolation [7]. **Reconstruction-based methods** usually apply sophisticated prior knowledge to narrow down the possible solution space, which can possibly reach an SR result of flexible and sharp details. However, the reconstruction-based methods are usually time-consuming and suffer from performance degradation when the scale factor increases. **Example-based methods**, also known as learning-based methods, generally utilize machine learning (ML) approaches to train a statistical model from substantial training examples and establish the mapping between a low-resolution (LR) image and the corresponding high-resolution (HR) one. In the area of example-based SISR methods, which benefited from the rapid development of deep learning (DL) techniques in recent years, various DL-based SISR methods have been explored and can usually achieve state-of-the-art performance. The research about DL-based SISR methods can be mainly categorized by **efficient neural network (NN) architectures** and **effective optimization objectives**.

2.1. Neural Network Architectures for SISR

Because of the ill-posed nature of SISR problems, it is of great importance to construct a high-quality NN architecture that performs upsampling from LR input to HR output. Generally, there are four kinds of network frameworks based on their upsampling operations and related layers' locations:
(1) **Pre-upsampling**: This framework uses pre-defined traditional upsampling algorithms to acquire preliminary HR representation and then refine it with the continuous neural network. An example is SRCNN proposed by Dong et al [5].
(2) **Post-upsampling**: Different from pre-upsampling, this framework replaces the pre-defined upsampling function with a learnable upsampling layer and puts it after the previous convolutional layers to save the computational resources. For example, Dong et al. [8] and Shi et al. [9] did the early works using this framework.
(3) **Progressive upsampling**: This framework utilizes a cascade of Convolutional Neural Networks (CNNs) to recon-

struct HR outputs progressively, which improves the shortcomings of learning difficulty in one step for the previous two frameworks and solves the need for multi-scale SR images. One classical implementation is the Laplacian pyramid SR network (LapSRN) [10].
(4)**Iterative up-and-down sampling**: This framework tries to compute the reconstruction error iteratively and then fuse it back to tune the output SR image. It could dig better the relationships between LR and HR image pairs and provide high-quality results. DBPN [11] proposed by Haris et al. is one of the classical models under this framework.
Based on these four general architectures, a couple of more advanced network design strategies have shown up. The following parts have listed some popular ones in recent years:
(1) **Residual learning**: With related research showing that the solution space of a DNN can be expanded by increasing its depth or width [12], residual CNN structure [13] was developed and applied within the NNs to broaden their depth and thus achieved better performances in DL-based SISR tasks [14].
(2) **Recursive learning**: By applying the same CNN modules repeatedly in a recursive manner, the design of recursive learning could help reduce the overall training parameters of the whole network and improve the reconstruction performance simultaneously. One classical implementation is DRCN proposed by Kim et al [15].
(3) **Dense connections**: With the purpose of acquiring richer information for reconstruction and restoring more high-quality details, dense connections fuse low-level and high-level features together by taking all the preceding layers' outputs as the current layer's inputs. For example, Tong et al. [16] proposed SRDenseNet, which construed dense blocks and built dense connections between different blocks.

2.2. Optimization Objectives for DL-based SISR Methods

In DL-based SISR, the choice of optimization objective is of great importance to the final performance. In the early stage, researchers mainly used the loss function of mean square error (MSE) per pixel. However, subsequent studies showed that the loss function could not sufficiently reflect the perceptual quality of the reconstructed image [17]. Thus, a series of alternative loss functions have been explored. Apart from MSE, a couple of popular loss functions are listed below:
(1) **Perceptual loss** [17]: Designed to evaluate the perceptual quality of images. It measures the high-level features' differences between SR and HR image pairs. In contrast with MSE which forcibly requests pixel matching, perceptual loss encourages the SR results to be perceptually similar to the target HR images.
(2) **Texture loss**: Designed for the goal that the reconstructed image should have the same style, such as colors, textures, and contrast with the target. For example, Sajjadi et al. [18] proposed EnhanceNet that utilized the texture loss calculated by Gram matrix of the VGG features of input data to produce visually more satisfactory results.
(3) **Adversarial Loss**: In recent years, Generative adversarial networks (GANs) have attracted more and more attention because of their strong ability to extract potentially hidden patterns from target HR images through the competition between the generator and the discriminator. In the training process of GANs, the adversarial losses related to the scores output by the discriminator are naturally used for more stable training and higher-quality results. Two of the classical GAN models for DL-based SISR are SRGAN [1] and ESRGAN [19].

3. DESCRIPTION OF MODELS

The goal of the SISR problem is to generate an SR image I^{SR} from an LR input image I^{LR} to estimate the high-resolution counterpart of the same scene, which is I^{HR}. This article aims to implement, explore and compare the performances of the SRResnet and SRGAN models inspired by Ledig et al.[1].

3.1. SRResnet

3.1.1. Residual Block

With related research showing that the deeper our network is, the larger potential solution space we have [12], deep Convolutional Neural Networks (CNNs) for SISR tasks have been explored. However, the traditional CNN architecture was proved to have a degrading performance when getting deeper because of **gradient explosion** and **gradient vanishing**. To solve this problem, He et al. [13] proposed the residual structure that makes it easier for every additional layer to contain the identity function as one of its elements. This results in the nested function classes, as Figure 2 shows.

The architecture of the basic residual block that constructs the SRResnet is shown in Figure 1. Assume $f(x)$ is the mapping we want, then the block only needs to learn the mapping of $f(x)-x$, which is proved easier in the training process, i.e., Gradients can be propagated lossless within the layers being deeper.

3.1.2. Network Structure

The structure of the SRResnet is shown in Figure 3. All the Conv layers of the network adopt appropriate padding to maintain the same widths and heights for the inputs and the outputs. The first Conv layer and the one after the upsampling blocks use the kernel size of 9 to perform the efficient feature extraction for the low-level features. And the kernel sizes of 3

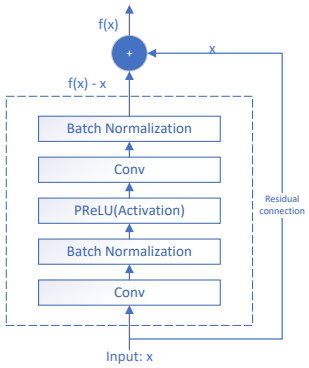

Fig. 1. Architecture of the residual block in SRResnet.

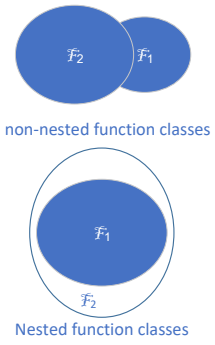

Fig. 2. Nested vs. non-nested classes.

are used for high-level features extraction. Batch Normalization (BN) layers follow the Conv layers to avoid the gradient explosion and gradient vanishing and help avoid overfitting.

For the activation layers, ParametricReLU (PReLU) is chosen rather than the popular ReLU activation function. Compared with ReLU which outputs zero for all negative inputs, PReLU sets a linear slope with a learnable parameter when $x < 0$, fixing the situation of "dying ReLU" where a traditional ReLU could be stuck on the negative side and always outputs 0. PReLU speeds up the training process.

After the preliminary convolution, it follows a sequence of B residual blocks, where B can be adjusted with any number. Specifically, a skip-connection that surrounds the residual blocks is used for better reconstruction. Skip-connection can provide more high-frequency information from high-level features, prevent gradient vanishing, and avoid re-learning redundant features.

Finally, the network uses the upsampling block consisting of Conv, pixel shuffle, and PReLU layers for image reconstruction from LR to SR image. As mentioned in Section 2.1, the location of the upsampling block follows the architecture of post-upsampling, which benefits from saving the computational resources and accelerating the reconstruction process. The pixel shuffler layer performs the upsampling process by Rearranging elements and transforming the tensor by reducing the number of channels and increasing the image widths and heights, from $(C * r^2, H, W)$ to $(C, H * r, W * r)$, where C, r, H, W are the number of channels, upscale factor, height, and width separately.

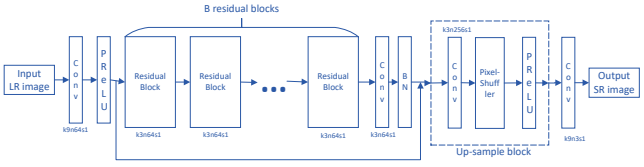

Fig. 3. Structure of the SRResnet. It is also used as the generator in the GAN network.

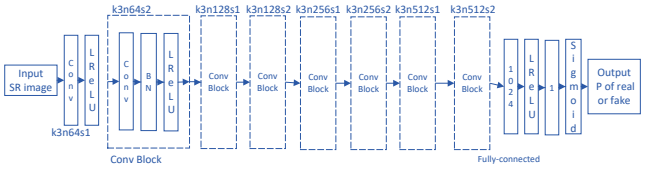

Fig. 4. Structure of the discriminator in the GAN network.

3.1.3. *Loss Function*

Denoting the SRResnet model as G, its parameters as θ_G, $\theta_G = W_{1:K}; b_{1:K}$ representing the weights and biases of a K-layer DNN, training HR images as I_n^{HR} and LR images as I_n^{LR}, loss function as l_{SR}, then the training goal is the optimized parameters $\hat{\theta}_G$, where:

$$\hat{\theta}_G = \arg\min_{\theta_G} \frac{1}{N} \sum_{n=1}^{N} l^{SR}\left(G_{\theta_G}\left(I_n^{LR}\right), I_n^{HR}\right) \quad (1)$$

And the loss function for SRResnet is the most popularly used pixel-wise loss, equal to Mean Square Error (MSE):

$$l_{MSE}^{SR} = \frac{1}{r^2 WH} \sum_{x=1}^{rW} \sum_{y=1}^{rH} \left(I_{x,y}^{HR} - G_{\theta_G}\left(I^{LR}\right)_{x,y}\right)^2 \quad (2)$$

where r is the upscale factor.

3.2. SRGAN

Although a high peak-signal-to-noise ratio (PSNR) the SRResnet could potentially reach, the reconstructed SR images usually lack high-frequency details and are perceptually unsatisfying and visually unreal. GANs provide an effective framework to improve the performance of reconstructing more natural images mainly by setting the adversarial loss that encourages the results to move closer to the natural image manifold. Generally, a GAN network trains a generative network G aiming to fool another discriminator D. In contrast, D is trained to distinguish SR images from HR ones. In this approach, G can be trained to generate the SR images that are similar to real ones and thus successfully fool the D model.

3.2.1. *Network Structure*

The SRGAN network contains two separate models, the generator G and the discriminator D. For G, the same network architecture as SRResnet mentioned in Section 3.1 is used. And the architecture of D is shown in Figure 4. Model D uses a series of Conv layers followed by BN and Leaky ReLU layers for the feature extraction process of the input SR image. Leaky ReLU can be seen as PReLU with the linear slope manually set in the negative input zone. The sequence of the Conv

layers is with the same kernel size of 3, increasing channels, and decreasing input resolution. The number of channels increases by double every two Conv layers, with the input size decreasing by double at the same time. Two fully-connected layers follow the Conv layers. And a final sigmoid function is set to obtain the probability of the classification between SR and HR images.

3.2.2. *Generator Loss Functions*

To avoid the situation that the results could be over smooth with the MSE loss solely, two additional kinds of losses were introduced with respect to perceptually relevant features, which are **content loss** and **adversarial loss**.

Content loss is defined as MSE loss between the feature maps output by a specific layer in a VGG [20] network, with the SR image and the real HR one as inputs separately. Denoting $\phi_{i,j}$ the feature map output by the activation layer after the j-th Conv layer and before the i-th max-pooling layer in a VGG19 net, then the content loss l_X^{SR} is:

$$\begin{aligned} l_X^{SR} &= l_{VGG/i,j}^{SR} \\ &= \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} \left(\phi_{i,j}\left(I^{SR}\right)_{x,y} - \phi_{i,j}\left(G_{\theta_G}\left(I^{LR}\right)\right)_{x,y} \right)^2 \end{aligned} \tag{3}$$

The **adversarial loss** of model G is designed according to the GAN framework to encourage the outputs on the natural image manifold. Because the optimal situation occurs when $D_{\theta_D}\left(G_{\theta_G}\left(I^{LR}\right)\right) = 1$, so the loss function can be written as:

$$l_{Gen}^{SR} = \sum_{n=1}^{N} -\log D_{\theta_D}\left(G_{\theta_G}\left(I^{LR}\right)\right) \tag{4}$$

Therefore, the total loss of G is defined as:

$$l^{SR} = \lambda_{mse} * l_{MSE}^{SR} + \lambda_{\text{perc}} * l_X^{SR} + \lambda_{\text{gen}} * l_{\text{Gen}}^{SR} \tag{5}$$

where the λs are the weights of each loss that are manually set.

3.2.3. *Discriminator Loss Function*

As the principle of the GAN framework, the discriminator D, along with model G, aims to solve the min-max problem of:

$$\begin{aligned} \min_{\theta_G} \max_{\theta_D} & \mathbb{E}_{I^{HR} \sim p_{\text{train}}\left(I^{HR}\right)} \left[\log D_{\theta_D}\left(I^{HR}\right) \right] + \\ & \mathbb{E}_{I^{LR} \sim p_G\left(I^{LR}\right)} \left[\log\left(1 - D_{\theta_D}\left(G_{\theta_G}\left(I^{LR}\right)\right)\right) \right] \end{aligned} \tag{6}$$

Therefore, the calculation of the loss l_D is done by Equation 6 when model G is fixed.

4. IMPLEMENTATION

4.1. Dataset Description

The dataset we use is DIV2K [21], which was used in NTIRE 2017 challenge on example-based single-image super-resolution. There are 900 high-resolution 2K RGB images of different contents publicly available in DIV2K. All the image files are stored in Portable Graphics Format (PNG) format. After rotating the images which have bigger heights than the corresponding widths to the form where the widths are bigger than the heights, an observation is reached that all the HR images in the dataset have the same width of 2040 but different heights. The minimal height of all the HR images is 648.

The corresponding LR images of the DIV2K HR ones are acquired according to the NTIRE 2017 challenge. There are two tracks, with the first track acquiring the LR images using the **bicubic** downsampling operator, and the second track using an **unknown** downsampling operator hidden by the organizer. For each track, there are 3 sub-tracks that downsample the original HR images by the factor of 2, 3, and 4 separately. Figures 5 and 6 are the samples of the LR images which are transformed from the original HR image by BicubicX4 and UnknownX4 downsampling separately.

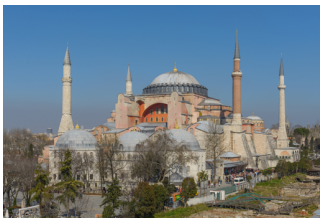

Fig. 5. LR image after BicubicX4 from 0890 DIV2K.

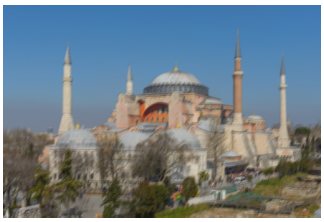

Fig. 6. LR image after UnknownX4 from 0890 DIV2K.

4.2. Dataset Split

For each sub-track as described in Section 4.1, there exist 900 LR-HR image pairs. Within each sub-track, we select the first 800 image pairs from the index of 1 to 800 as the training data, 50 pairs from 801 to 850 as the validating data, and the last 50 pairs from 851 to 900 as the testing data.

4.3. Data Pre-Processing and Enhancement

For the more efficient training, the input data values are normalized from the pixel value range of [0, 255] to the range of [0,1]. Moreover, to control the computational resource occupancy when training and provide more variable input data to feed the model, a sub-area of $3 * 180 * 180$ is randomly cropped from the original image. Besides the **random cropping** as one of the data enhancement tricks, another way used is to **flip the cropped image horizontally, vertically**, or **rotate it by 90 degrees clockwise or counter-clockwise** separately with an independent probability of 0.2. The implementation of the mechanism mainly utilized the Python package **opencv-python**.

4.4. Evaluation Metrics

It is of great importance to choose the appropriate quantitative metrics to access the model performance on the validating and testing data. One of the most popular criteria is the peak-signal-to-noise ratio (**PSNR**), which is entirely linearly correlated to the MSE function. The range of PSNR values is $[0, +\infty]$, and the higher PSNR values indicate better performance. However, PSNR focuses on the pixel-by-pixel comparison and is short of the overall perceptual quality of the reconstructed image. Apart from PSNR, another frequently used metric is the structural similarity method (**SSIM**). Unlike PSNR, SSIM evaluates the image distortion degree from the combination of loss of correlation, luminance distortion, and contrast distortion. Therefore, we choose to use PSNR and SSIM to quantitively evaluate the model performance in the validating and the testing stages. The implementation of these two metrics is with the help of the packages **torch** and **pytorch_msssim**. Moreover, to access the image texture quality more accurately, we calculate the metrics on the images that are transformed from the RGB channel to the YCbCr channel when validating and testing.

4.5. Training Details and Parameters of SRResnet

As stated in Sections 4.1, 4.2, and 4.3, we use the augmented training data and train 6 different versions of SRResnet within the 6 different sub-tracks. The implementation of the whole training process from data loading to the SR images output mainly utilizes the Python packages of **torch**, **numpy**, and **opencv-python**. As the measurement of the computational efficiency, the numbers of the **trainable parameters** of SRResnets are shown in Table 1. For a faster gradient descending, we set a small training data **batch size** of 8. Moreover, the training process uses **ADAM optimizer** by setting the **learning rate** of 10^{-4} and other functional parameters by default. Furthermore, a **scheduler** is set to decrease the learning rate by 90% every 30 epochs.

The model structure implemented is shown in Figure 3 in Section 3.1, with the number of residual blocks set as 16. The loss function used when training the network is MSE, as stated by Equation 2.

Training convergence occurs when the training loss value is minimized and oscillates in a small range within the continuous epochs in the training process. And the validating metrics values usually present similar trends as the training loss. To avoid the problem that the model may be overfitted and have decreasing validating metrics values, an **early stopping** mechanism should be applied. In this implementation, there is a maximum number of training epochs set as 300. Moreover, an early stopping criterion is set: A "patient" value is set as 30, and the best validating PSNR during the training process is recorded. If there is no new best validating PSNR occurring to replace the last best one within the patience of 30 epochs, we stop training in advance. Since the model training curves and validating curves with different downsampling operators have similar trends, we only put the curves within "BicubicX4" here. The pixel loss curve in the SRResnet training process with "BicubicX4" is shown in Figure 7, and the validating metrics value curves are shown in Figure 11. The vertical lines mark the epoch that reaches the highest validating PSNR. As we can see, the SRResnet training pixel loss, validating PSNR, and validating SSIM have reached a stable zone without severe changes after the vertical lines. The trends of validating PSNR and SSIM are similar, increasing simultaneously during the model training process.

Upscale factor	2	3	4
SRResnet/ G in SRGAN	1401749	1586389	1549462
D in SRGAN	80188609	80188609	80188609

Table 1. The number of the trainable parameters of SRResnet / SRGAN Generator, and SRGAN Discriminator under different upscale factors.

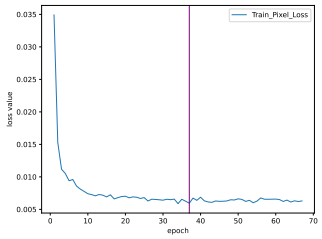

Fig. 7. SRResnet pixel training loss, BicubicX4.

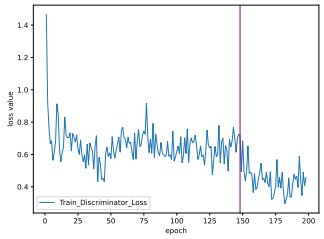

Fig. 8. SRGAN Discriminator training loss, BicubicX4.

4.6. Training Details and Parameters of SRGAN

Implementation of SRGANs uses the same datasets as SRResnets in all the sub-tracks and utilizes the same Python packages. The numbers of their **trainable parameters** are shown in Table 1. SRGAN would take more time to train due to the high number of parameters of the discriminator. Moreover, the parameter setting of **batch size** and **optimizer** of SRGANs follows the same as those of SRResnets. However, GANs networks are generally harder to train compared to CNNs. Therefore, we set the **learning rate** of 10^{-4} and a **scheduler** to decrease the learning rate by 90% after 150 epochs. And the early stopping patience is set at 50 epochs.

The generator model structure in SRGAN uses the same as SRResnet. And the discriminator's structure is shown in Figure 3.1.2, in which the use of Leaky ReLU rather than PReLU is for reducing the trainable parameters and speeding up the training. Even though the same structures of the generator and SRResnet, they have different loss functions. The loss functions of the generator and the discriminator are calculated by Equations 5 and 6 separately. The weights of the

pixel loss, content loss, and adversarial loss that combine the generator loss are $\lambda_{mse} = 2$, $\lambda_{\text{perc}} = 1$, and $\lambda_{\text{gen}} = 0.005$.

The training loss curves of the generator and the discriminator, and validating metrics curves of SRGAN within "BicubicX4" are shown in Figures 9, 10, 8, and 12. We separate the content loss and other generator losses within two figures because the content loss operates in a relatively small numerical. Specifically, as we can see from Figures 9 and 10, the three kinds of weighted losses sum up to the generator total loss. Pixel loss and content loss tend to change in a small value range after the vertical line. However, the adversarial loss of the generator and the discriminator loss still maintain the trends that are relatively unstable after the vertical line. This is because of the nature of the GAN framework. The better performance the generator has, the higher discriminator loss and possible adversarial loss could show up. Because of this nature, the GANs are generally hard to train, and we can expect a better model performance by extending the training epochs and other fine-tuning tricks.

As Figure 12 shows, the validating PSNR and SSIM tend to change severely even after the vertical line that marks the best epoch of PSNR. From this observation, we can find that PSNR and SSIM metrics are not objective and accurate to measure the performance of SRGAN which aims to generate the photo-realistic SR images, proving the statements of Ledig et al. [1] and Wang et al. [19]. This is to say, the gradient descending of SRGAN is not completely related to PSNR and SSIM, different from the PSNR-oriented SRResnet.

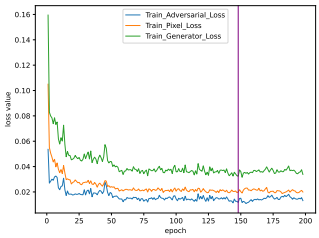

Fig. 9. Pixel loss, adversarial loss, and total loss of SRGAN generator, BicubicX4.

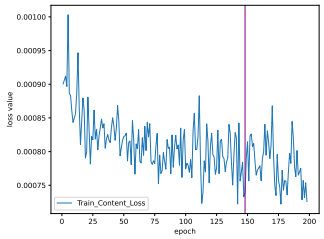

Fig. 10. Training content loss of SRGAN BicubicX4.

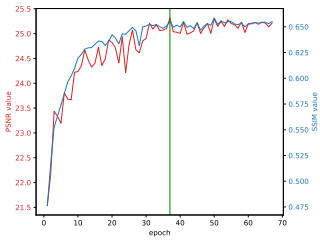

Fig. 11. SRResnet validating PSNR and SSIM, BicubicX4.

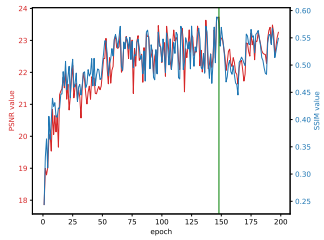

Fig. 12. SRGAN validating PSNR and SSIM, BicubicX4.

5. EXPERIMENTAL RESULTS AND ANALYSIS

5.1. Results on Testing Datasets

To test the performance of SRResnet and SRGAN, we use the last 50 LR-HR image pairs with the picture indices $851 - 900$ in each sub-track as the testing datasets and calculate the average PSNR and SSIM values. The results are shown in Table 2. Moreover, the visualized reconstruction results of picture 0890 in the testing dataset are shown in Figure 13.

	BicubicX2	BicubicX3	BicubicX4	UnknownX2	UnknownX3	UnknownX4
PSNR_SRResnet(dB)	29.84	26.90	24.81	26.84	26.00	23.05
PSNR_SRGAN(dB)	26.78	24.25	23.55	25.37	23.44	22.21
SSIM_SRResnet	0.8569	0.7307	0.6337	0.7300	0.685	0.5486
SSIM_SRGAN	0.7764	0.6107	0.5627	0.6659	0.5728	0.4903

Table 2. Average testing PSNR and SSIM of SRResnet and SRGAN with different downsampling operators.

Fig. 13. SR Reconstruction results with different downsampling operators of the test image 0890 in DIV2K.

5.2. Discussion

By observing the testing results in Table 2, we can find that the overall performance on PSNR and SSIM of SRResnet is better than that of SRGAN. However, from the visualization

results in Figure 13, we find that the SRResnet results are over smoothed. And the results of SRGAN perform better texture quality and high-frequency details, making the reconstructed images more realistic. Moreover, the results present the overall higher PSNR and SSIM values and visual performance under the bicubic downsampling operator than the unknown operator. Although unknown, we can have a general impression from the comparison of Figures 5 and 6: The image ghosting using the unknown operator is more severe and is more blurred than using the bicubic. Besides, the reconstruction performances of both models degrade with the upscale factor going up, becoming visually more blurred, showing more artifacts. And SR images of SRGAN are showing mixed colors, such as in BicubicX3 and UnknownX3.

5.3. Future Improvements

Although SRGAN possesses the ability to reconstruct the LR image with more realistic textures, there still exists a gap between the reconstructed image and the real one. One phenomenon we can find is that there exist some blurry details accompanied by artifacts as discussed in Section 5.2. Therefore, we will discuss the approaches to improve the SRGAN performance from the perspectives of optimizing the network structure, optimizing loss functions, and other improvements.

5.3.1. Network Structure Improvement

There are a couple of approaches to strengthen the SRGAN network structure according to the related research. Firstly, as stated by Tong et al. [16], the use of dense connections could improve the learning ability of deep networks, at the same time avoiding over-fitting and accelerating the model training. Therefore, an executable approach for this SRGAN can be introducing the dense connection between the residual blocks of SRResnet, which is also the generator in SRGAN. Specifically, Wang et al. [19] proposed using Residual-in-Residual Dense Block (RRDB) to improve the network capacity of SRGAN and reached the good results of better-recovered textures and reduced unpleasing noises.

Besides the dense connections, Wang et al. [19] also empirically found that the removal of BN layers could better deblur and increase the network performance. BN layers are trained to acquire the mean and standard variance values during the training process and use them to normalize the input images. However, due to the ill-posed nature of SISR tasks, the testing datasets could possibly possess different mean and variance values, thus disturbing the reconstruction results. The experiments done by Wang et al. have shown the effect of fewer artifacts in the recovered images.

5.3.2. Loss Function Improvement

In the current SRGAN, there are multiple kinds of losses combined for the generator model training. As stated in Section 3.2, the losses of the generator are pixel-wise MSE loss, content loss, and adversarial loss. The content loss is calculated by the MSE between high-level feature maps of the SR and HR image proceeded by a trained VGG-19 network. However, a more efficient feature extractor such as a deeper VGG network or a trained Resnet could possibly provide a better feature mapping and thus improve the texture quality on the reconstructed images. Besides, the calculation of the adversarial loss is based on the discriminator's output which estimates the probability of the input image as a real or a fake one. And as proposed by Wang et al. [19], this can be improved by changing the output of the discriminator to calculate the probability of how many degrees a real image is relatively more realistic than a fake one. With this approach, the generator can benefit from a more efficient gradient descending. Moreover, with experiments by Wang et al., this modification helps SRGAN learn sharper edges and more realistic textures.

Besides better constructing the existing losses, we can further introduce more related losses in model training. From comparing of the results between SRResnet and SRGAN, we can find the deficiency solely using the pixel-based MSE loss. Therefore, more kinds of losses that help improve the perceptual quality of the recovered image can be explored, such as the texture loss [18] stated in Section 2.2.

5.3.3. Other Improvements

We find two possible aspects to improve from the experiments. The first is to focus more on image denoising. From the comparison of bicubic and unknown samples in Figures 5 and 6, we notice the need for adding denoising functions in the data pre-processing or as a trainable layer in the network. This could help improve the performance, especially for the unknown upsampling operator. Secondly, as discussed in the previous sections, we find the metrics of PSNR and SSIM cannot accurately reflect the SR image quality concerning the human visual system. Thus, new metrics such as the Mean Opinion Score (MOS) should be used.

6. CONCLUSION

This article has explored the principles, advantages, and disadvantages of SRGAN and SRResnet. Then, we implement these two models on the DIV2K dataset, with bicubic and unknown downsampling operators and the upscale factors from 2 to 4. Results show SRGAN's stronger ability to generate realistic results than SRResnet, which tends to output oversmoothed SR images. Moreover, both models' performances degrade with upscale factors growing up. Finally, limitations and future improvements with aspects of model structure, loss functions, metrics, and denoising are discussed.

7. REFERENCES

[1] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al., "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4681–4690.

[2] Dengxin Dai, Yujian Wang, Yuhua Chen, and Luc Van Gool, "Is image super-resolution helpful for other vision tasks?," in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2016, pp. 1–9.

[3] Hayit Greenspan, "Super-resolution in medical imaging," *The computer journal*, vol. 52, no. 1, pp. 43–63, 2009.

[4] Tao Lu, Jiaming Wang, Yanduo Zhang, Zhongyuan Wang, and Junjun Jiang, "Satellite image super-resolution via multi-scale residual deep neural network," *Remote Sensing*, vol. 11, no. 13, pp. 1588, 2019.

[5] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang, "Learning a deep convolutional network for image super-resolution," in *European conference on computer vision*. Springer, 2014, pp. 184–199.

[6] Dong Cheng and Kit Ian Kou, "Fft multichannel interpolation and application to image super-resolution," *Signal Processing*, vol. 162, pp. 21–34, 2019.

[7] Robert Keys, "Cubic convolution interpolation for digital image processing," *IEEE transactions on acoustics, speech, and signal processing*, vol. 29, no. 6, pp. 1153–1160, 1981.

[8] Chao Dong, Chen Change Loy, and Xiaoou Tang, "Accelerating the super-resolution convolutional neural network," in *European conference on computer vision*. Springer, 2016, pp. 391–407.

[9] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1874–1883.

[10] Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang, "Deep laplacian pyramid networks for fast and accurate super-resolution," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 624–632.

[11] Muhammad Haris, Gregory Shakhnarovich, and Norimichi Ukita, "Deep back-projection networks for super-resolution," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 1664–1673.

[12] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio, "On the number of linear regions of deep neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[14] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee, "Enhanced deep residual networks for single image super-resolution," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2017, pp. 136–144.

[15] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee, "Deeply-recursive convolutional network for image super-resolution," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1637–1645.

[16] Tong Tong, Gen Li, Xiejie Liu, and Qinquan Gao, "Image super-resolution using dense skip connections," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 4799–4807.

[17] Justin Johnson, Alexandre Alahi, and Li Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *European conference on computer vision*. Springer, 2016, pp. 694–711.

[18] Mehdi SM Sajjadi, Bernhard Scholkopf, and Michael Hirsch, "Enhancenet: Single image super-resolution through automated texture synthesis," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 4491–4500.

[19] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy, "Esrgan: Enhanced super-resolution generative adversarial networks," in *Proceedings of the European conference on computer vision (ECCV) workshops*, 2018, pp. 0–0.

[20] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[21] Eirikur Agustsson and Radu Timofte, "Ntire 2017 challenge on single image super-resolution: Dataset and study," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.