Explore Super-Resolution Deep Learning Methods

Anonymous cvpr submission

Paper ID *****

Abstract

Super-resolution is one important and interesting problem in computer vision, which takes low-resolution images and outputs those with visually higher resolution. Scholars have advanced dozens of super-resolution techniques to tackle this problem, including traditional techniques and deep learning models. In this final project, we'd run and reimplement one traditional technique Bicubic Interpolation and two deep learning methods SRCNN and SRGAN over one common dataset DIV2K dataset to compare them in terms of accuracy and efficiency.

1. Introduction

Super-resolution takes low-resolution images and outputs those with visually higher resolution. While these output images have very high quality visually, it is much less expensive compared with obtaining originally high-resolution images. Here are some concrete scenarios where higher-quality images may be beneficial: medical images with a clearer view of details may help for diagnosis; satellite imagery with higher resolution could provide more information and routing guidances; video surveillance (super-resolution could be generalized from static images to videos if the model could run fast enough) would help more in action recognition and person identification; and more. At the same time, obtaining high-quality images from the very beginning is usually expensive due to equipment costs. Given these, super-resolution techniques would be very useful if their accuracy and efficiency are promising, with the former measured by the difference between generated super-resolved images and high-resolution ground truth and the latter one in terms of training resources and super-resolution processing time per image.

Super-resolution has been coped with from different perspectives. Traditional methods include but are not limited to BiLinear, BiCubic, and BiCubic Spline Interpolations. While Linear Interpolation tries to fit a straight line (a polynomial with order 1) between adjacent points for $1D$ case, BiLinear Interpolation copes with $2D$ case. BiCubic Interpolation tries to calculate a polynomial with order 3, and BiCubic Spline Interpolation intakes the information provided by derivatives additionally. In general, these methods are based on mathematical modelings, which are relatively simple with the underlying at most three order polynomials. BiCubic Interpolation usually serves as the benchmark technique for super-resolution.

Deep learning has been unprecedentedly successful these years. Those deep learning techniques from computer vision also play a role in solving super-resolution tasks, and some work shows visually amazing super-resolved images. Two from these are Super-Resolution Convolutional Neural Network (SRCNN) and Super-Resolution Generative Adversarial Network (SRGAN) [2, 3]. More technique details of these two will be given in 2.2 and 2.3.

In this final project, we'd run and reimplement Bicubic Interpolation, SRCNN, and SRGAN (explained more deeply in 2) over one common dataset DIV2K dataset, evaluate and compare them using metrics defined in 3.2. They are not all successful as planned and what has been done would be described in 4.

2. Approach

2.1. Bicubic Interpolation

Bicubic Interpolation is one of the traditional techniques using usual math modeling and computations.

In the $1D$ case, given (x_i, y_i) pairs, to approximate the y value at $0 < x < 1$, you could fit a 3-order polynomial to approximate the ground truth $f : x_i \mapsto y_i$ over interval $(0, 1)$. 3-order polynomials have 4 parameters and using data pairs at $x \in \{-1, 0, 1, 2\}$ gives four equations, producing one unique set of 4 parameters. Figure 1 visually illustrates this idea.

This idea and computation could generalize from $1D$ to $2D$. With $2D$ discrete points, you can first interpolate them into several lines and then interpolate those lines into a $2D$ plane. Figure 2 may give some visual intuition.

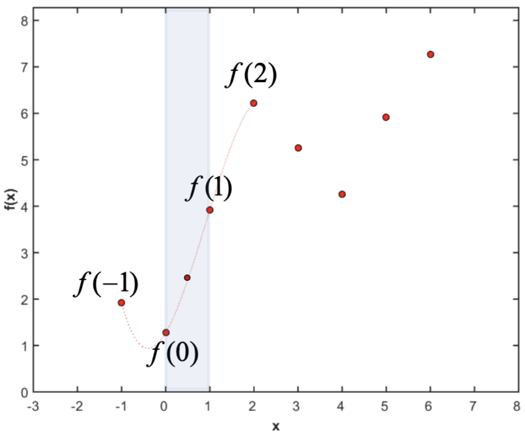


Figure 1. Illustration for $1D$ Cubic Interpolation. Source: https://www.mssc.mu.edu/~daniel/pubs/RoweTalkMSCS_BiCubic.pdf

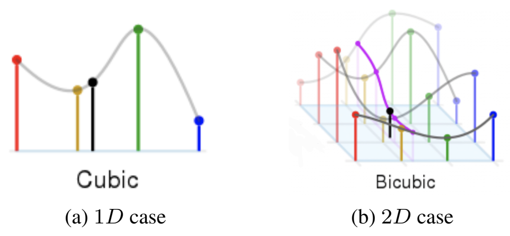


Figure 2. Source: https://en.wikipedia.org/wiki/Bicubic_interpolation

2.2. SRCNN

SRCNN is a deep learning method invented by Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang [2]. SRCNN is featured with fast training due to its simple architecture of three convolutional layers (approximately, training of eight hundred images can be accomplished in one minutes on average with GPU) and good performance in terms of common super-resolution metrics. This End-to-End super-resolution technique directly maps low-resolution images to high ones. Specifically, after preprocessing images by bicubic interpolation for upscaling, three operations compose the overall mapping $F = F_3$ (F maps from low-resolution image Y to its high-resolution one X):

1) Patch extraction and representation F_1 :

$$F_1(Y) = \max(0, W_1 * Y + B_1)$$

2) Non-linear mapping F_2 :

$$F_2(Y) = \max(0, W_2 * F_1(Y) + B_2)$$

3) Reconstruction $F_3 = F$

$$F(Y) = F_3(Y) = W_3 * F_2(Y) + B_3$$

where each operation is accomplished by the basic convolution layers with W_i, B_i and ReLU (except the last one). With the goal of approximating X by $F(Y)$, the loss is chosen to be mean squared error (MSE) loss.

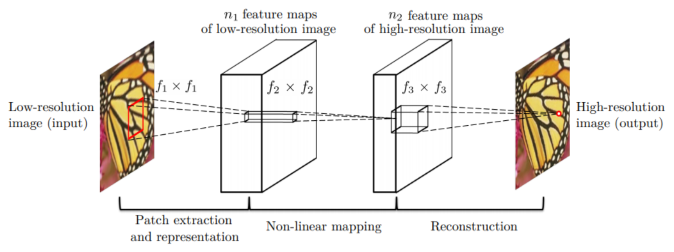

n_1 feature maps of low-resolution image
n_2 feature maps of high-resolution image
Low-resolution image (input)
$f_1 \times f_1$
$f_2 \times f_2$
$f_3 \times f_3$
High-resolution image (output)
Patch extraction and representation
Non-linear mapping
Reconstruction

Figure 3. Source of image: [2]

To explain the process of our experiment, we train SRCNN from scratch with Adam optimizer over 640 pairs of images and use 160 image pairs as the validation dataset. Each pair contains a high-resolution image from DIV2K dataset and its small-sized counterpart. Due to limited computation resources and the trend that loss and psnr do not improve significantly after around epoch 15, early stopping before all epochs (49) finish may be reasonable. Also, this may help to avoid over-fitting.

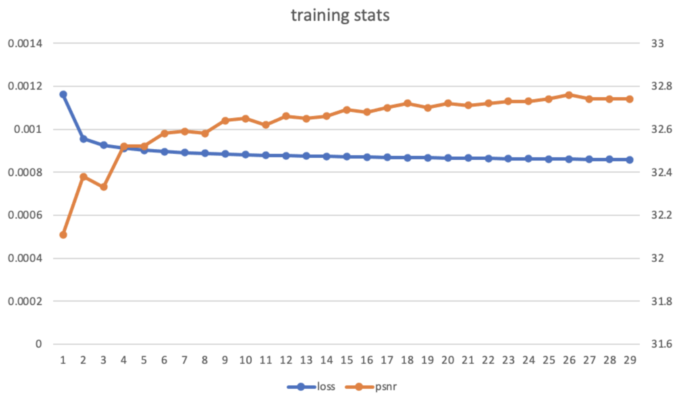


Figure 4. Training statistics: loss and psnr over epochs

2.3. SRGAN

The generative adversarial network (GAN) model was first proposed by Ian Goodfellow et al. [4]. The original GAN architecture is comprised of a generator model for producing synthetic outputs and a discriminator model that classifies each sample from a mix of ground truth and synthetic examples as real or fake. The two models are trained simultaneously in an adversarial process. While the discriminator is trying to classify real and fake samples as accurately as possible, the generator is learning to produce plausible samples to fool the discriminator. When the generator's task is specified to output images with high resolution, then this model becomes SRGAN, proposed by [3].

The architecture is very very complex compared with SRCNN introduced before. For bigger figure 5 for SRGAN's architecture visualization, please refer to the source. Here are some verbal summaries of it. The Generator Network uses convolutions, Parametric ReLU, Batch Normalization, Elementwise Sum, and Pixel Shuffler following some specific combinations where the kernel size k, number of feature maps n, and stride s are denoted in the image. Note that this network also applies skip connections across each of its five residual blocks locally and globally. This may help to remember information and avoid the vanishing gradient problem. The Discriminator Network is relatively simple. It's composed of convolutions, Leaky ReLu, and Batch Normalization following some patterns. Then, dense and sigmoid layers follow to produce binary classification being real or fake.

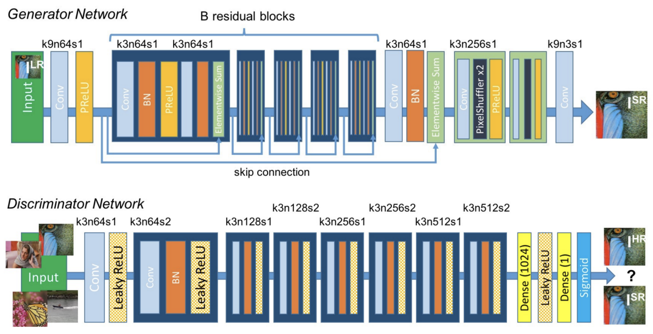


Figure 5. Source: [3]

GAN is an adversarial min-max problem, which means that the generator wants to minimize the following loss while the discriminator wants to maximize it:

$$\min_{\theta_G}\max_{\theta_D}\mathbb{E}_{I^{HR}\sim p_{train}(I^{HR})}[\log D_{\theta_\theta}(I^{HR})]+ \\ \mathbb{E}_{I^{LR}\sim p_G(I^{LR})}[\log(1-D_{\theta_D}(G_{\theta_G}(I^{LR})))]$$

This loss is also very complex and let's break it into components. D is the discriminator network parameterized by θ_D, and G is the generator parameterized by θ_G. I^{HR} is real high-resolution image, while $G_{\theta_G}(I^{LR})$ is super-resolved one from low-resolution images. D wants to give higher score for the former one and lower score for the latter case, which makes both parts of the loss high. While the discriminator is trained to distinguish real or fake high-resolution images, the generator wants to fake the discriminator. Thus, the generator's optimization goal is the opposite.

While the loss above is the general one for GAN framework/idea, the concrete (perceptual) loss for SRGAN is the following, whose assessment is more relevant to perceptual characteristics compared to normal MSE:

$$l^{SR} = l^{SR}_{content\_loss} + \frac{1}{1000} l^{SR}_{adversarial\_loss}$$

$$l^{SR}_{content\_loss,VGG/i.j} = \frac{1}{W_{i,j}H_{i,j}}\sum_{x=1}^{W_{i,j}}\sum_{y=1}^{H_{i,j}} \\ (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y})^2,$$

where $\phi_{i,j}$ denotes a specific feature map from pretrained 19 layer VGG network, and we can understand this loss as composing $\phi_{i,j}$ before computing MSE loss.

$$l^{SR}_{adversarial\_loss} = \sum_{n=1}^{N} -\log D_{\theta_D}(G_{\theta_G}(I^{LR}))$$

For the training procedure, it is trained over the same dataset as SRCNN and starts from the pre-trained model VGG19. However, these training and reimplementation tasks for SRGAN are **not fully accomplished** due to some difficulties and would be discussed in 4.

3. Experiments

3.1. Dataset

We choose DIV2K dataset dataset to be our common dataset [1]. According to [5], it contains 1000 images with large number of pixels (average number of pixels 2, 793, 250) and very high quality.

3.2. Metrics

We mainly use peak signal-to-noise ratio (PSNR) as our metric. Even though it does not capture visual perceptions well, it is easy to calculate, standard, and popular. Its specific formula is:

$$PSNR = 10\log_{10}(\frac{L^2}{MSE(I,\hat{I})}),$$

where I is the ground truth image, $\hat{I}$ is the super-resolved one, and L is the maximum pixel value (255 in our case).

3.3. Qualitative Results

Due to page limit, we give the following images

Visually, the one generated by Bicubic Interpolation is smoother while the one by SRGNN is more rigid, or has more clear edges.

3.4. Quantative Results

Accuracy: The average PSNR over 10 selected test images is 32.4067 for SRCNN and 31.0666 for bicubic interpolation.

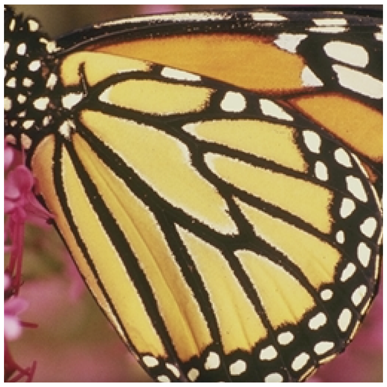


Figure 6. Butterfly Ground Truth

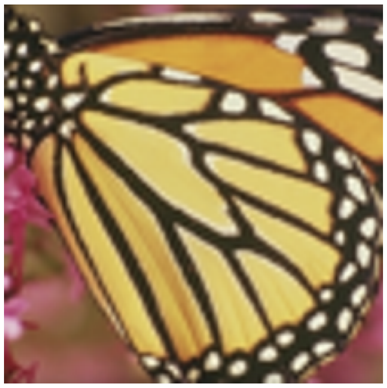


Figure 7. Butterfly Bicubic Interpolation

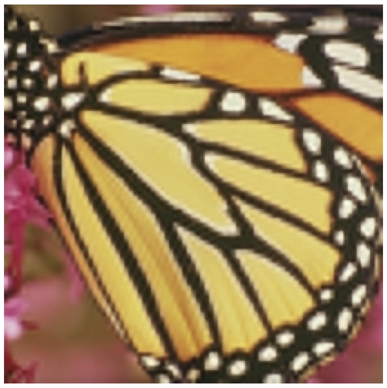


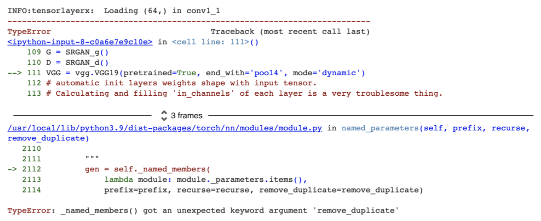
Figure 8. Butterfly SRCNN

Efficiency: SRCNN takes about 1min per epoch on average for training with GPU; the trained model super-resolves every image with about 0.1 second on averge. In comparison, Bicubic Interpolation does not need training; it takes about 2 minutes 17 seconds to process each image on average.

4. Implementation

For bicubic interpolation and SRCNN, their open-source codes are first ran and then we reimplement parts, ex. bicubic interpolation's calculation, layers' declaration and model class for SRCNN.

For SRGAN, it's trained halfway for the first time and could not proceed because the computation resource became unavailable then. It's moved to Colab but it got stuck on the following error. According to some online posts, it might be related to outdated packages, but the problem is not solved with all package updated.



```
INFO:tensorlayerx:  Loading (64,) in conv1_1
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-8-c0a6e7e9c10e> in <cell line: 111>()
    109 G = SRGAN_g()
    110 D = SRGAN_d()
--> 111 VGG = vgg.VGG19(pretrained=True, end_with='pool4', mode='dynamic')
    112 # automatic init layers weights shape with input tensor.
    113 # Calculating and filling 'in_channels' of each layer is a very troublesome thing.

                         3 frames
/usr/local/lib/python3.9/dist-packages/torch/nn/modules/module.py in named_parameters(self, prefix, recurse,
remove_duplicate)
   2110
   2111         """
-> 2112         gen = self._named_members(
   2113             lambda module: module._parameters.items(),
   2114             prefix=prefix, recurse=recurse, remove_duplicate=remove_duplicate)

TypeError: _named_members() got an unexpected keyword argument 'remove_duplicate'
```

References

[1] 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. 3

[2] Kaiming He Chao Dong, Chen Change Loy and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2016. 1, 2

[3] Ferenc Huszar Jose Caballero Andrew Cunningham Alejandro Acosta-Andrew Aitken Alykhan Tejani Johannes Totz Zehan Wang Wenzhe Shi Christian Ledig, Lucas Theis. Photo-realistic single image super-resolution using a generative adversarial network, 2016. 1, 3

[4] Mehdi Mirza Bing Xu David Warde-Farley Sherjil Ozair Aaron Courville Yoshua Bengio Ian J Goodfellow, Jean Pouget-Abadie. Generative adversarial networks. *Advances in neural information processing systems*, pages 2672–2680, 2014. 2

[5] Steven C.H. Hoi Zhihao Wang, Jian Chen. Deep learning for image super-resolution: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(10):3365–3387, 2021. 3