EE 541 – Computational Introduction to Deep Learning

Image Super Resolution

Lujia Zhong, Jingxiang Zhang

December 12, 2022

Abstract

Purpose: This is a super-resolution (SR) project. We propose to recover high-resolution images from low-resolution images, using the dataset of DIV2K. Image super-resolution is a critical step to recover and even enhance the information transmitted by the image. It has many important applications in other fields, such as satellite image remote sensing, and digital high-definition. There are many computer vision solutions to this problem, including interpolation-based methods, reconstruction-based methods, and learning-based methods.

We will first test various SR models to enhance the quality of given images that are common in the natural environment, using Peak signal to noise ratio (PSNR) and structural index similarity (SSIM) as criteria for model performance, and compare the deep learning models with Bicubic and other interpolation algorithms. Finally, we will test our best model on the whole training set and analyze the result.

In the extension work, we will compare how the patch (sub-image) size and batch influence the training process. Then we design an experiment to show that training Y channel only is reasonable and generally better than training RGB channels together. At last, we will compare how the upscale factor influences the pre-upsampling model.

Model: SRCNN, FSRCNN, ESPCN, VDSR, SRGAN, EDSR, DCSCN, SUBCNN, Per. SRGAN

1. Introduction

1.1 Motivation

SR refers to the process of using optics and its related optical knowledge to restore image details based on known image information. It is to increase the resolution of the image and prevent its image quality from deteriorating. SR technique has important application value in the fields of monitoring equipment, satellite image remote sensing, digital high-definition, microscopic imaging, video coding communication, video restoration, and medical imaging. There are two significant meanings for super-resolution: On the one hand, it can be applied to the real world, solving the problems of insufficient equipment, insufficient data sources, and limited network bandwidth. On the other hand, it can also help to solve related problems, such as deblurring, and denoising, by migrating the intermediate feature maps of the super-resolution model to other deep neural network models.

In the photography field, The higher the resolution, the clearer the imaging can be obtained. At the same time, higher resolution also means larger storage space. For mobile devices with very

limited space, especially for the old mobile device, there will be a trade-off between resolution and storage space. But with SR, it can restore the low-resolution images taken 10 years ago. The SR algorithm also implements in the mobile phone picture browsing module. The display resolution of the same picture on different mobile phones varies because of the different SR models in use. With the deep learning model starting to appear, it can increase at least 30% percent of resolution for the mobile phone camera. And SR technique is all around the Image Signal Processing (ISP) pipeline. From the moment the camera shutter is pressed, all the following steps: color calibration, tone mapping, multi-frame noise reduction, etc, are related to the SR technique.

In the network video transmission field, SR also has a great contribution. When people watch video programs, they prefer to use a higher resolution. Taking Youtube videos as an example, it provides multiple resolution configurations, from 144p to 1080p. However, due to the network bandwidth, not everyone can play watch 1080p all the time, especially on the cellular network. Due to the SR technique, now it can transmit low-resolution video at first, and then implement real-time resolution improvement at the user's end. This is valuable in live video, video play, and download applications.

In addition to being a separate task, the image SR technique can also be used as pre-processing of other image work or as an independent small module. It is significant in many fields, such as improving the resolution of small targets in target detection tasks to improve monitoring systems and improving the recognition of fuzzy targets in medical images. Nowadays, image super-resolution technology is used in the restoration of many precious historical photos and videos, which has great humanistic value and commemorative significance.

1.2 criteria

There are two criteria in this project used to compare the performance of each model, Peak signal to noise ratio (PSNR) and structural index similarity (SSIM).

i. PSNR

PSNR is used to measure the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of a signal. Since many signals have a very wide dynamic range, PSNR is often used as a logarithmic quantity by decibel scale. PSNR is mainly used in the image processing field to quantify the reconstruction quality of images and videos affected by lossy compression. In this project, we will use PSNR to measure how close the image that is processed by our model and the original high-resolution image is. The PSNR formula is given by:

$$PSNR = 10 \times \log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20 \times \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \tag{1}$$

where MAX is the maximum pixel value, which is equal to 255 for unit8 type of input data. Mean Square Error (MSE) is:

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\|I(i,j) - K(i,j)\|^2 \tag{2}$$

where I(i,j) and K(i,j) stand for the pixel value in i row and j column in the two images.

This criterion is only used to calculate single-channel images. For multi-channel images, additional processing methods are required. Hence we only compare the Y channel result, this criterion can be used directly.

ii. SSIM

SSIM measures the similarity between two images. The theory of SSIM is that humans are not sensitive to the absolute brightness/color of pixels, but are very sensitive to the position of edges and textures. SSIM mimics human perception by focusing primarily on edge and texture similarities. We divide the two images into small patches. Then we compare the images patch by patch. Given a patch 'x' from one image and a corresponding patch 'y' from another image, we compute the following summary statistics for pixel intensities in each patch.

$$l(x,y) = \frac{2\mu_x\mu_y}{\mu_x^2+\mu_y^2} \quad c(x,y) = \frac{2\sigma_x\sigma_y}{\sigma_x^2+\sigma_y^2} \quad s(x,y) = \frac{\sigma_{xy}}{\sigma_x\sigma_y} \tag{3}$$

Where l is the luminance similarity, c is the contrast similarity, and s is the structural similarity. μ_x and μ_y is the mean of x and y, σ_x and σ_y is the variance of x and y. And σ_{xy} is the covariance of x and y. Finally, We multiply them together, and add some extra constants to avoid division by zero errors:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2+\mu_y^2+c_1)(2\sigma_x^2+\sigma_y^2+c_2)} \tag{4}$$

2. Literature Review

There are three categories of Super-resolution methods: interpolation-based methods, reconstruction-based methods, and learning-based methods (i.e. deep learning methods). Intuitively, it's easy to think that interpolation could be a powerful non-deep-learning technique to upscale images. However, these types of methods, including nearest-neighbor, bicubic, Sinc and Lanczos, etc. are usually easy to implement and generate not satisfying outcomes. One of the reasons is that all of them only use local information from a single image to increase its resolution, which means that it's not learnable. After that, people began to use deep learning-based methods to solve this problem, and lots of frameworks and structures designed for SR emerged. There are four major frameworks: pre-upsampling SR, post-upsampling SR, progressive upsampling SR, and iterative up-and-down sampling SR.

i. Pre-upsampling SR

For pre-upsampling SR, the key idea is that it is difficult to learn the map between low-resolution images and high-resolution images. Therefore, it first upsamples LR by some algorithm. The upsampled image size is the same as HR, and then the model learns their mapping relationship, which greatly reduces the difficulty of learning.

In 2016, Dong et al. proposed SRCNN [1], a representative of the pre-upsampling method, to map interpolated LR images to HR images, where use interpolation functions as the pre-upsampling method. This model only use a few convolution kernels to map the image. At the same time, J Kim et al proposed VDSR [2], using deep CNN to do feature mapping. In the next year, DRRN [3] was proposed by Ying et al. to use a Recursive Residual Network.

However, Pre-upsampling will amplify the noise and make the image blurring, and the pre-processing algorithm is performed in a high-dimensional space, which requires more time and space costs.

ii. Post-upsampling SR

The difference for this method is that post-upsampling SR computing in the original low-resolution image can improve efficiency. And then perform upsampling operations at the end of the network. Since the feature extraction process with huge computational costs is only used in low-dimensional space, it greatly reduces the computation time and space complexity. Also, the up-sampling layer is learnable instead of a fixed interpolation algorithm to make this operation feasible. This framework has also become one of the most mainstream frameworks.

Also in 2016, Chao et al. proposed FSRCNN [4], using the deconvolution layer as the final out layer to increase the size of an image. Shi et al. proposed ESPCN [5], using another strategy in the final out layer, which he called Efficient Sub-pixel Convolution.

The drawback of Post-upsampling SR is that, for a large-scale factor, the model is difficult to train.

iii. Progressive up-sampling SR

When the scale factor is large, it is very difficult to use the method above. And for different scale factors, different SR network model needs to be trained separately, which cannot meet the needs of multi-scale SR. The key idea of this method is that it uses a cascaded CNN structure. High-resolution images are gradually reconstructed layer by layer. It breaks down difficult tasks to reduce learning difficulty. If you want to do 4 times SR, you need to do 2 times SR first, then do 4 times SR on the basis of X2_SR. A representative of this is the Laplacian pyramid SR network (LapSRN) proposed by Lai et al. [6]. This type of framework can satisfy different needs for up-scaling factors.

But the model design is complex, and the training is unstable. It needs more modeling guidance and advanced training strategies.

iv. Iterative up-and-down sampling SR

Iterative up-and-down sampling SR methods reconstruct LR and HR images alternately across the network and generate final HR images from all these intermediate reconstructions. This method can understand the relationship between LR-HR image pairs better, which leads to better reconstruction results. A representative of this framework is DBPN, which is proposed by Haris et al. in 2018 [7]. to explore the mutual dependency of LR and HR pairs better. However, the reverse projection design criteria are unclear, with great potential for exploration.

3. Implementation

3.1 Data

The dataset being used is from DIV2K: https://data.vision.ee.ethz.ch/cvl/DIV2K/, which originally consists of 800, 100, and 100 images for training, validation, and testing, respectively. Because test data is not provided from the official website, training data has to be split into two parts for training and validation, and the original validation data is for testing. Overall, the data being used contains 550 training images, 250 validation images, and 100 test images in png format (total of 4.72 GB), the categories include environment, flora, fauna, people, scenery, etc. Each image is RGB and is not in the same size of pixels, across from size (inputs images) of (558, 1020) and (1020, 324) to (1020, 1020). The HR labels will be 2x the size of the LR inputs.

3.2 Pre-processing

There are three obvious challenges:

1. Large dataset: It's impossible to load all the data into GPU memory altogether while training.

2. Varying size: Training images appear to be of different sizes, patching or clipping is needed for multi-batch training.

3. Large model: A few models with different structures and sizes are trained on the same training data, which means some large models cannot even be trained with one single image in its original size.

Hence, to deal with the above problems, training data is split into 50×50 patches for main work (Sec. 3.5), which is achieved by patching images to their smallest size that is divisible by 50 (Ex. 615 or 649 to 650; 1020 to 1050). We end up with training data with total of 237405 patches. The validation and test data are not split and will be used one by one (batch size equal to 1) when inference.

Additionally, all RGB images are converted to YCbCr color space, and only Y will be input into the network. The metrics of SSIM and PSNR are calculated on Y as well. For visualization of images, Cb and Cr will be interpolated to 2X of the original size and combined with predicted Y, then converted to RGB color space.

3.3 Model

In this work, 3 interpolation-based baselines and 9 deep-learning models are chosen to compare.

- Baseline: Interpolation with Bicubic, Bilinear, and Nearest
- Pre-processing model: One pre-processing model of VDSR is used with Bicubic interpolation for up-sampling.
- Post-processing model: Eight post-processing models are trained, including up-sampling methods of Deconvolution and Sub-Pixel.

Two of the post-processing models are GAN-based: SRGAN and Perceptual SRGAN. The second one is with the same structure as SRGAN except for training with an additional perceptual loss with VGG19. Perceptual loss is proven by experiment to be useful for training stability of the GAN model.

There are two main upsampling methods for the post-processing model: deconvolution and sub-pixel. Both of them increase the resolution of an image. Deconvolution uses a deconvolutional layer, which is a type of layer in a neural network that can be used to increase the resolution of an image. Sub-pixel convolution, on the other hand, uses a type of convolutional layer that essentially rearranges the pixels in channel dimension to the pixels in the image (Ex. rearranging 1×4×2×2 image with 4 channels to 1×1×4×4 image with 1 channel but 2x size). It's difficult to say which method is better overall, as it will depend on the specific circumstances and application. Hence, in this work, both of them are adopted.

3.4 Loss Function

The loss function being used is mainly MSE loss, except for L1 loss for the EDSR model. For the GAN model, Mean Squared Error (MSE) loss is used for Generator and Binary Cross-Entropy

(BCE) Loss for Discriminator. The reason for choosing MSE loss is that it's differentiable and a good choice for regression task, including this SR problem. However, the L1 loss is adopted in its original work, because it "provides better convergence than L2" [9].
All performance results are coming from calculating SSIM and PSNR on Y (in YCbCr) of the target image and predict image. YCbCr is a color space that separates the luminance (Y) and chrominance (Cb and Cr) components of an image. Predicting Y instead of RGB allows less computation and predicted value. And luminance component contains the most important visual information in an image, which makes it enough for models to generate a better high-resolution image than using RGB color space.

3.5 Model Selection and Comparison

In this part, all models are trained on 550 training images and test on 250 validation images. The training paramters are kept the same for them: learning rate is 0.001, batchsize is 100, and epoch of training is 30.

Table 1: SR model with corresponding methodology and performance on DIV2K Validation. Interpo. and Pre. in Framework column represent interpolation and pre-processing, respectively. Deconv. in Upsampling column represents deconvolution. The Inference column indicates the image's average inference time in seconds for one image in the validation set.

Model	Framework	Upsampling	PSNR	SSIM	Inference (s/image)
Bicubic	Interpo.	Bicubic	**31.1991** ± 4.8922	**0.9384 ± 0.0436**	0.2360
Bilinear	Interpo.	Bilinear	30.6582 ± 4.8207	0.9308 ± 0.0486	0.2325
Nearest	Interpo.	Nearest	30.2609 ± **4.6585**	0.9289 ± 0.0474	**0.2229**
SRCNN var. [10]	Post	Sub-Pixel	33.2217 ± 5.0683	0.9582 ± 0.0317	0.2282
FSRCNN [11]	Post	Deconv.	33.1202 ± 5.0646	0.9576 ± 0.0320	0.2303
ESPCN [12]	Post	Sub-Pixel	33.4330 ± 5.0781	0.9597 ± 0.0311	**0.2273**
VDSR [13]	Pre.	Bicubic	**33.6951** ± 5.0855	**0.9612 ± 0.0304**	0.2914
SRGAN [14]	Post	Sub-Pixel	**33.6539** ± 5.0178	**0.9611 ± 0.0304**	0.4292
EDSR [15]	Post	Sub-Pixel	31.4187 ± 4.8169	0.9421 ± 0.0405	0.2760
DCSCN [16]	Post	Deconv.	32.7570 ± 5.1077	0.9545 ± 0.0343	0.2599
SUBCNN	Post	Sub-Pixel	**33.6304** ± 5.1129	**0.9610 ± 0.0305**	0.2315
Per. SRGAN	Post	Sub-Pixel	32.1676 ± **4.3690**	0.9484 ± 0.0373	0.4288

As shown in Table 1, all deep learning models generate better performance than interpolation baseline models according to metrics of PNSR and SSIM, where the best-performed model is VDSR, a model based on the pre-processing framework and used bicubic interpolation for upsampling. Additionally, the most stable model while training is VDSR as well, and the most unstable model is SRGAN. This is understandable for GAN because the instability of GAN while training is a challenging problem since it's been invented. The reason for VDSR to be stable in the training process may partially be because it's a pre-processing model, which means the whole model structure only inputs and outputs an image with a higher resolution but keeps its size unchanged. This process may make it become much more stable than other models when training.

According to the last column, the deep learning model with the fastest inference speed is ESPCN, only taking $\tilde{0}.2273$ s for one image inference, which is even faster than the interpolation with

the Bicubic and Bilinear methods.

The model of Per. SRGAN is a variant of SRGAN proposed by ourselves. The only change is adding perceptual loss to SRGAN's generator model. After using this loss, the SRGAN model training process became more stable even though didn't show better performance. And the standard deviation of PNSR became smaller. The training stability can be seen in Figure 1.

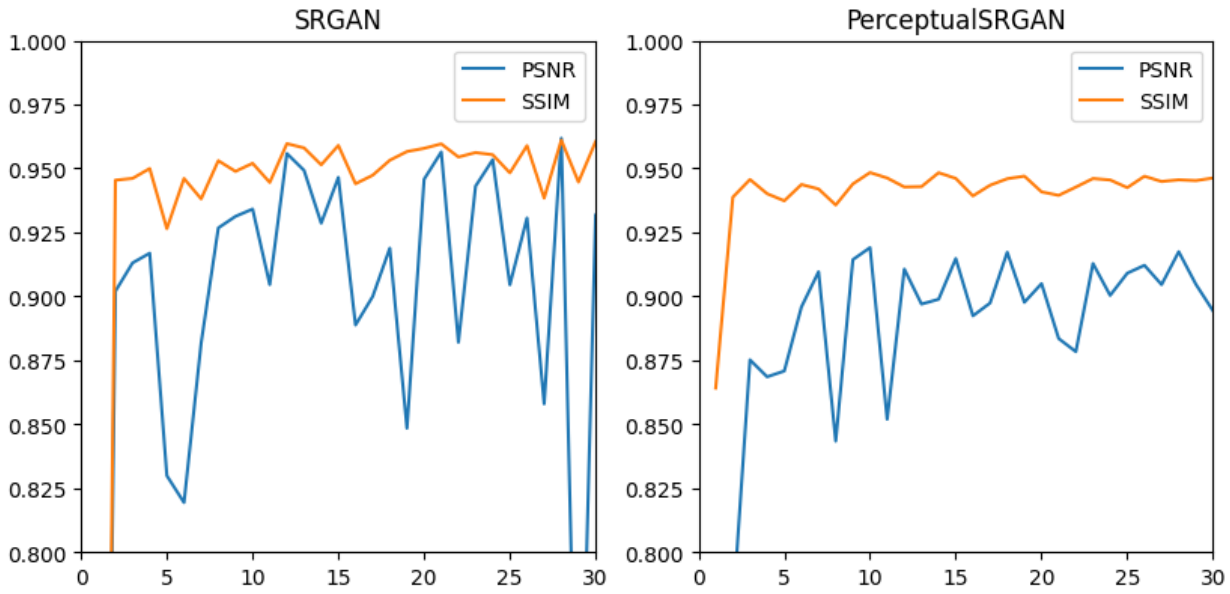

Figure 1: Training process of SRGAN and Perceptual SRGAN. The PSNR values are divided by a constant to scale to the same range of SSIM for better visualization.

A visual comparison between models can be seen in Figure 2. It's not easy to distinguish between a few well-performed models such as SRGAN, SUB, VDSR, etc. But it can be clearly seen that they are all much better than traditional interpolation methods and some poor-performed models like EDSR and DCSCN.

4. Result and Analysis

4.1 Test Set Result

According to the PNSR and SSIM metrics on validation data, the model with the best performance and has a good inference speed is VDSR. Then this model is chosen to be retrained on the whole training and validation images for 30 epochs, and then test it on test images. The final result on the test images is shown in Table 2, which is higher than its corresponding result in Table 1 because more data (validation images) is provided.

Table 2: VDSR performance on final test images.

Model	PSNR	SSIM
VDSR	33.7546 ± 4.9174	0.9614 ± 0.0299

Figure 2: Training process of SRGAN and Perceptual SRGAN.

4.2 Analysis

In total four images (predictions), two with the highest PSNR and two with the lowest PSNR are selected for visualization. A heatmap visualization as shown in Figure 3 is conducted for further validation and explanation of which parts of the images contribute the most to low PSNR and SSIM. In the first line are the original target images, and the second line is the predicted HR images by the final VDSR model with corresponding heatmaps added. The redder the map is, the lower the PNSR/SSIM is in that region. It's obvious that The first two images with the highest metric scores contain relatively less information. The background in the first two images is blurred by either heavy fog or camera blurring technique, which makes these two images have less information. But the other two images are much high-resolution and contain many details including grass and coniferous leaves, which make it much more challenging to do super-resolution on them. This can be validated by the heat map in the second line. There is little red heat in the first two images. In the last two images, the red heat is mainly focusing on the ground with grass and leaves on the trees.

5. Extension

After doing research on the Super Resolution network performance area, there are still something we wonder, and these fields are rearly explored in other research.

1. How does patch size (sub-image) influence the training?
2. How does the channel influence the training?
3. How does the scale factor influence the pre-upscaling model?

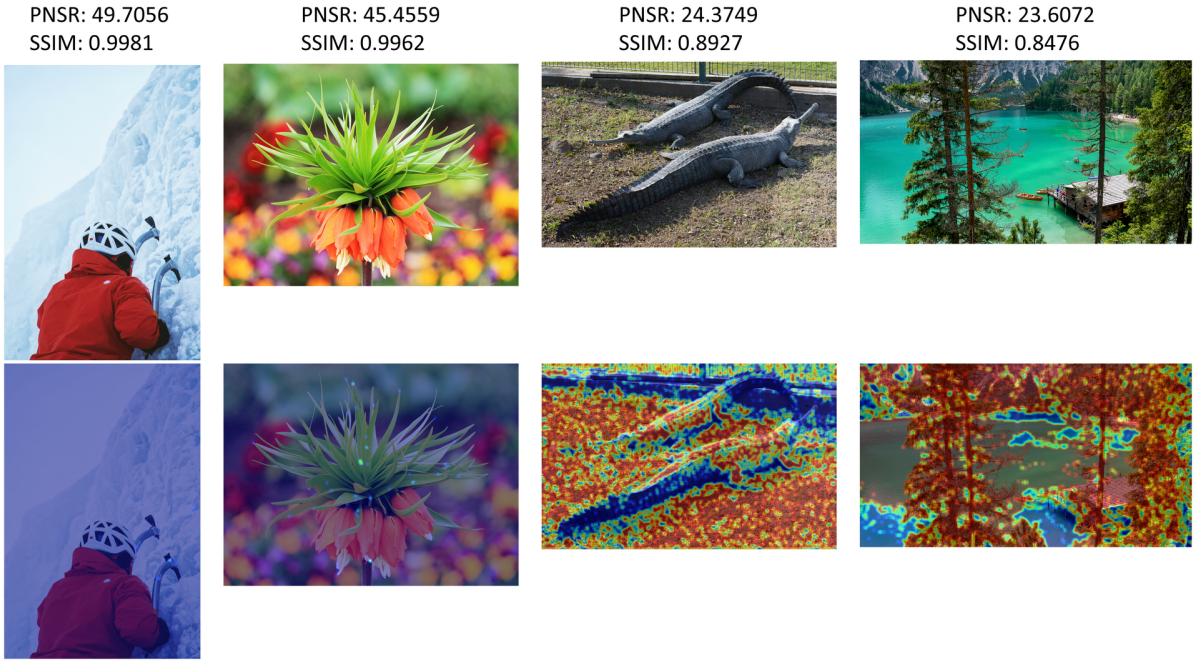

Figure 3: Visualization of four images and corresponding heatmaps.

Next, we will use some experiments to demonstrate these three pieces of research.

5.1 Patches Size

In most of the super-resolution network models, the default image pre-processing method is to cut the training image into small patches and train the network by those small patches. In [1], Dong et al. proposed using 33*33 as a sub-image size with a stride equal to 14. Also in [8], Zhou, et al. show that the smaller the stride, the smaller the MSE, and they use a patch size range from 3*3 to 13*13. In our previous experiment, we use a patch size equal to 50. These default settings aroused our curiosity, does the sub-image patch size influence the training process? Hence we design these experiments to compare the training process given different patch sizes.

We will use the ESPCN model since it is a relatively simple model, using MSE loss as a loss function. Training the model by Adam optimizer with learning rate decay by 0.9 for each epoch, and start with 0.01. The batch size of training is 16. We cut the training images into 32*32, 64*64, 128*128, and 256*256 with no overlapping and padding with 0 in the image border (instead of dropping it). Also, we convert the RGB channel into the YCbCr channel and only train and compare the performance on the Y channel as with previous settings. Here, I will train the model for 20 epochs. 20 epochs may not converge completely, and may not reach the best performance yet, but our goal is only to make comparison between the training process.

Figure 4 shows that when the patch size is equal to 64, the model will get the best performance. A patch size equal to 32 is also a good choice, but the smaller the patch size, the longer it will take to train. When the patch size is larger than 64, the larger the patch size, the worse the performance is.

However, one should notice that for a smaller patch size, it has more iterations for each epoch, i.e., the data input into the model at each iteration is different. The smaller patch size has the

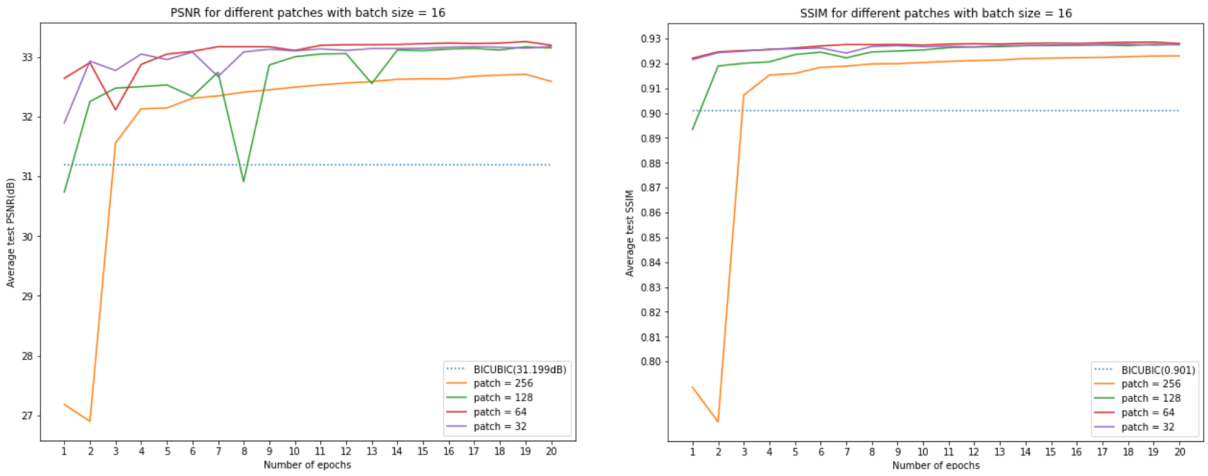

Figure 4: training process for different batch size

advantage that it can train more iterations for each epoch. To make sure that the data size input into the model has the same size, we design another group of training. For patch sizes equal to 32, 64, 128, and 256, we use the batch size = 1024, 256, 64, 16. This time, the input data have the same size, and the iteration for each epoch are close to each other (because we pad the image when cutting, the larger the sub-image, the larger blank will be padding, therefore the iteration here will not equal).

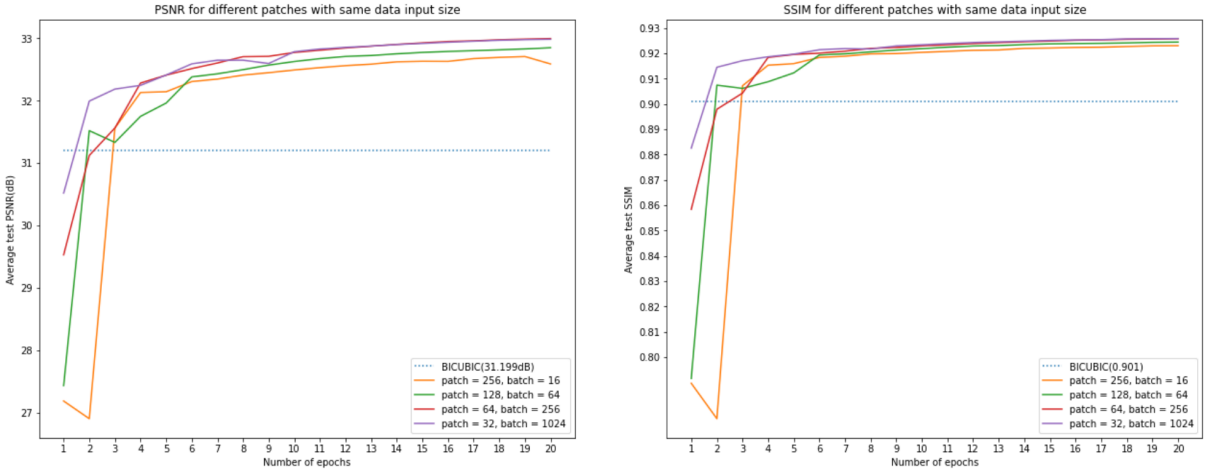

Figure 5: training process for the same data size

Figure 5 shows that even if we keep the same input data size, a patch size equal to 32 or 64 still gets the best performance, and also, the larger the patch size, the worse the performance. The table below shows the final test performance on the validation set.

Table 3: ESPCN model with the different batch size and patch size

id	batch_size	patch_size	total_iterations	PSNR(dB)	SSIM
1	16	32	482360	33.1725 ± 5.0636	0.9276 ± 0.0468
2	16	64	121280	**33.1949** ± 5.0673	**0.9280** ± 0.0468
3	16	128	32500	33.1496 ± 5.0734	0.9275 ± 0.0470
4	16	256	8340	32.5883 ± 4.8995	0.9230 ± 0.0483
5	64	128	8120	32.8484 ± 5.0599	0.9244 ± 0.0479
6	256	64	7580	32.9948 ± 5.0709	0.9258 ± 0.0476
7	1024	32	7520	32.9840 ± 5.0715	0.9258 ± 0.0477

Conclusion for this part: Table 3 shows that, when batch size equals 16, and patch size equals 64, the model will get the best performance. However, one should notice that it will spend more time to train more iterations. The training time between all these batch sizes and patch size settings maybe varies model by model. Given the same training time id = 2 spend, id = 6 can train more epochs, and performance may finally exceed id = 2.

5.2 Training Channel

In almost all the super-resolution research, the original image will convert into a YCbCr channel, and only the Y channel will be used to train the model. In YCbCr color space, Y refers to the luminance component, Cb refers to the blue chroma component, and Cr refers to the red chroma component. Since human vision is more sensitive to brightness changes than to chromatic changes. In this part, we will try to validate that training the Y channel only is reasonable.

We will still use the ESPCN model in this part, using MSE loss as the loss function. Training the model by Adam optimizer with learning rate decay by 0.9 for each epoch, and start with 0.01. The batch size of training is 64, and the patch (sub-image) size is 64*64. We will train the model for 20 epochs, and record PSNR for each epoch, SSIM will not be used.

For the first model, we will train the Y channel only. First, we read the low-resolution image as LR and convert the image to YCbCr, extracting Y channel as LR_Y. Then we make Bicubic interpolation on LR and get LR_After, convert LR_After to YCbCr, extract Cb and Cr channel as LR_After_Cb and LR_After_Cr. We put LR_Y as ESPCN model input, and get LR_After_Y. Finally, concatenate LR_After_Y, LR_After_Cb, and LR_After_Cr together, and convert back to LR_After_RGB. Then we read the high-resolution image as HR_RGB and convert the image to YCbCr, extracting the Y channel as HR_Y. We will calculate the PSNR of LR_Y and HR_Y as the first pair, and LR_After_RGB and HR_RGB as the second pair. In our expectation, the PSNR of the first pair will be higher than the second pair, since this model is trained on the Y channel. But the PSNR of LR_After_RGB and HR_RGB will not be too low, because the Y channel is the denominate channel.

For the second model, we will train the RGB channel together, the input channel of the model is 3, and the output is also 3. PSNR is calculated by RGB channels together.

For the third model, we will train the R, G, and B channel separately, each model train only one channel, and calculate PSNR on its own channel. Finally, we take the average PSNR as the model performance.

$$AVE(PSNR_R, PSNR_G, PSNR_B) = \frac{10}{3} \times (\log_{10}\left(\frac{MAX_I^2}{MSE_R}\right) + \log_{10}\left(\frac{MAX_I^2}{MSE_G}\right) + \log_{10}\left(\frac{MAX_I^2}{MSE_B}\right))$$
$$= 10 \times \log_{10}\left(\frac{MAX_I^2}{\sqrt[3]{MSE_R \times MSE_G \times MSE_B}}\right) \tag{5}$$

However, the real RGB channel PSNR should be calculated in the next formula. But the arithmetic mean and the geometric mean of $MSE_R + MSE_G + MSE_B$ are very close in this case. We use the previous formula as PSNR performance.

$$PSNR_{RGB} = 10 \times \log_{10}\left(\frac{MAX_I^2}{MSE_{RGB}}\right) = 10 \times \log_{10}\left(\frac{MAX_I^2}{\frac{1}{3}(MSE_R + MSE_G + MSE_B)}\right) \tag{6}$$

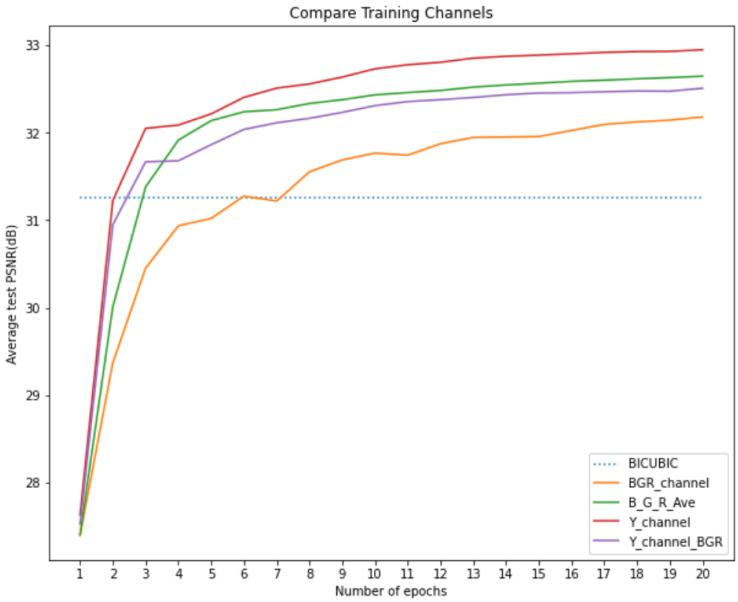

Figure 6: training process for different channels

Figure 6 shows the training curve for the ESPCN model using different training channels. Y_channel stands for the model training on the Y channel, and the PSNR is for Y channel comparison, while Y_channel_BGR stands for the model training on the Y channel, and the PSNR is for RGB channel comparison (here we use BGR because OpenCV read the image by BGR channel). BGR_channel stand for the model training on RGB channel together, and B_G_R_Ave stand for the average PSNR for 3 models training on R, G, and B channel separately.

From Figure 6, we can find that the worst model is training on the BGR channel together. While training on B, G, and R separately is only slightly better than training on the Y channel only (compare to the PSNR on the BGR channel). Given enough epochs, they may finally reach the same performance. However, training three models will spend more time. Therefore, training the Y channel only is very reasonable.

Figure 7 show the performance on the test image. All the models are better than the Bicubic one. According to this figure, it is difficult to tell the difference between these three models. However,

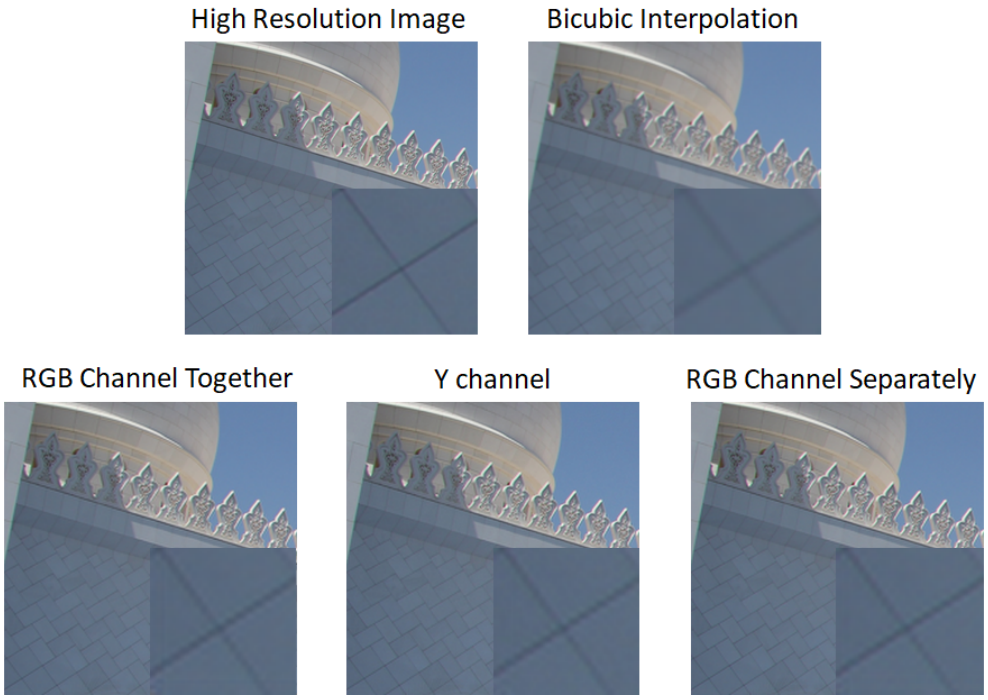

Figure 7: Model Performance for Training Different Channels

one can spot that all three images below have ripples around the black line. Since ESPCN uses a convolutional layer, like many filters, the black color in the black line will influence the surrounding area, that's why there are ripples around the black line.

5.3 Upscale Factor

In the Pre-upsampling SR algorithm, the algorithm will first upsample the image by Bicubic interpolation, then input the image into the model. Our training image is a low-resolution image, and the label is a high-resolution image. The model itself is to find a map between the low-resolution image and the high-resolution image. So, we wonder, if the input training image is a very low-resolution image, then the model must be more capable to restore a low-resolution image to a high-resolution image.

Hence, we design this part to verify our thought. For all the training images, we downsample the image by a factor of n and then upsample the image by a factor of n, and the new image will be used as a low-resolution training image. This dataset is marked as Data_Xn. In this part, we will use the 2, 3, 4, and 5 as scale factors, then we have Data_X2, Data_X3, Data_X4, and Data_X5 datasets. Use SRCNN as a model, the training hyper-parameter is the same as the previous settings. Therefore, we will train 4 models by these 4 datasets, and name them SRCNN_X2, SRCNN_X3, SRCNN_X4, SRCNN_X5.

In [1], Dong et al. compare the PSNR and SSIM pair by pair, which means, use SRCNN_X2 to test the performance Data_X2, use SRCNN_X3 test the performance Data_X3, and so forth. In our expectation, SRCNN_X5 has the best capability of restoring the image, then it should outperform SRCNN_X2 even in Data_X2. Hence, we will use each model to test each dataset, which will be 4 $\times$ 4 = 16 PSNR and SSIM results, and find out which model is the best for each dataset.

The other application of this part is that suppose we have an SRCNN_X2 pre-upsampling model,

and we want to upscale an image by 3 times, can we use this model? Or shall we need to train a new SRCNN_X3 model? If the SRCNN_X2 model performs approximately equal to SRCNN_X3 on Data_X3, then we don't need to train a new one.

Table 4: PSNR for Pre-upsampling Model with Different Scale Factor

Model	Data_X2	Data_X3	Data_X4	Data_X5
SRCNN_X2	31.2133	27.3398	25.8277	24.5237
SRCNN_X3	29.8539	27.3564	25.9380	24.6498
SRCNN_X4	28.4133	27.0070	25.8789	24.6790
SRCNN_X5	27.0745	26.3647	25.6352	24.6672

Table 5: SSIM for Pre-upsampling Model with Different Scale Factor

Model	Data_X2	Data_X3	Data_X4	Data_X5
SRCNN_X2	0.8980	0.8135	0.7615	0.7143
SRCNN_X3	0.8658	0.8042	0.7579	0.7136
SRCNN_X4	0.8217	0.7842	0.7491	0.7106
SRCNN_X5	0.7735	0.7538	0.7316	0.7024

Table 4 and 5 shows the PSNR and SSIM for different scale datasets and models. Compare SRCNN_X2 and SRCNN_X3 model, we can find that SRCNN_X2 outperform SRCNN_X3 in Data_X2, and SRCNN_X3 slightly outperform SRCNN_X2 in Data_X3, which mean we don't need to train a new model for X3 images if we have SRCNN_X2. And SRCNN_X3 also outperform SRCNN_X2 in Data_X4 and Data_X5, which meet our expectation since SRCNN_X3 is more fitful to map a more vague image into the high-resolution image.

Generally, the most fitful model for a given dataset is its corresponding upscale model. However, SRCNN_X4 and SRCNN_X5 perform worse. The convolutional kernel size in SRCNN is 9, 1, 5, and the receptive field in this model is 13. Perhaps a 13*13 receptive field is not enough to infer the high-resolution image for a very low-resolution image input, maybe a deeper network is needed to handle this.

6. Conclusion and Discussion

In this SR project, we first compare the PSNR, SSIM, and Inference speed for different models, and the result shows that VDSR, SRGAN, and SUBCNN have the best PSNR and SSIM performance for the given dataset and hyperparameters. ESPCN has the best inference speed and the PSNR and SSIM performance is next to the best 3 models. Then we compare how the patch size and batch influence the training process, and it shows that a relatively small patch size is more fitful for training SR models. Then we design an experiment to show that training Y channel only is reasonable and generally better than training RGB channels together. Finally, we find that the upscale factor will influence the pre-upsampling model, and the best model for a given upscale factor is its corresponding upscale model.

There are some questions that we feel "answered" after this project.

1. Previously, we used a convolutional neural network to do image classification. Use a convolutional layer to do feature extraction. But in this project, we have a deeper understanding of how convolution work. It is not only feature extraction, but also feature mapping. It uses the locality in image space and can do whatever we want it to do.

2. We used to believe that for a neural network, "deep is better" and "complicated is better". However, in this SR project, ESPCN is a very simple model and easy to train, but it still gets very good performance. While DCSCN is a complicated model with dozens of layers and similar techniques used in ESPCN, it performs worse.

3. The implementation of the loss function is more flexible than we believe before. MSE loss function can not only be used in function regression but also in any model that we want to fit for something. In this case, we use MSE for the convolutional layer output and the high-resolution image.

There is one thing that we are still curious about.

1. The reason SR models work is that they can increase the information of low-resolution images. But this is no more than "guess". Is there any limitation to this "guess"? Are there any relations between the information of a low-resolution image and the information of its corresponding high-resolution "guessed" image? For example, for an original image that has 100 percent of the information, if we compress it to 80 percent of the information, what is the information of this image if we use the SR models to recover this image? What about we compress it to 50 percent or 30 percent of the information? This may help us find a tradeoff between the image compression rate and the recovery effect.

There are still works we can do in the future.

1. Implement these SR models as a tool. Maybe provide super-resolution API in our personal website, help other people that want to use SR in their project.

2. Explore on image denoising field. Transfer the convolutional layer of SR models to image-denoising models.

3. Build the SR models and their corresponding network parameters in the hardware, like FPGA. Perform super-resolution on the real-time video output.

4. Applying different augmentation methods including cropping, flipping, scaling, rotation, color jittering, or randomly shuffling RGB channels.

References

References

[1] Dong, C. , et al. "Image Super-Resolution Using Deep Convolutional Networks." IEEE Trans Pattern Anal Mach Intell 38.2(2016):295-307.

[2] Kim, J. , J. K. Lee , and K. M. Lee . "Accurate Image Super-Resolution Using Very Deep Convolutional Networks." IEEE Conference on Computer Vision & Pattern Recognition IEEE, 2016.

[3] Ying, T. , Y. Jian , and X. Liu . "Image Super-Resolution via Deep Recursive Residual Network." IEEE Conference on Computer Vision & Pattern Recognition IEEE, 2017.

[4] Chao, D. , C. L. Chen , and X. Tang . "Accelerating the Super-Resolution Convolutional Neural Network." European Conference on Computer Vision Springer, Cham, 2016.

[5] Shi, W. , et al. "Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network." IEEE, 10.1109/CVPR.2016.207. 2016.

[6] Lai, W. , et al. "Fast and Accurate Image Super-Resolution with Deep Laplacian Pyramid Networks. " IEEE transactions on pattern analysis and machine intelligence 41.11(2019):2599-2613.

[7] Haris, M. , G. Shakhnarovich , and N. Ukita . "Deep Back-Projection Networks For Super-Resolution." arXiv (2018).

[8] Zhou, Dongsheng, Wang, et al. Depth Image Super Resolution Based on Edge-Guided Method[J]. Applied Sciences, 2018.

[9] Lim, Bee, et al. "Enhanced deep residual networks for single image super-resolution." Proceedings of the IEEE conference on computer vision and pattern recognition workshops. 2017.

[10] Dong, Chao, et al. "Learning a deep convolutional network for image super-resolution." European conference on computer vision. Springer, Cham, 2014.

[11] Dong, Chao, Chen Change Loy, and Xiaoou Tang. "Accelerating the super-resolution convolutional neural network." European conference on computer vision. Springer, Cham, 2016.

[12] Caballero, Jose, et al. "Real-time video super-resolution with spatio-temporal networks and motion compensation." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[13] Kim, Jiwon, Jung Kwon Lee, and Kyoung Mu Lee. "Accurate image super-resolution using very deep convolutional networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[14] Ledig, Christian, et al. "Photo-realistic single image super-resolution using a generative adversarial network." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[15] Lim, Bee, et al. "Enhanced deep residual networks for single image super-resolution." Proceedings of the IEEE conference on computer vision and pattern recognition workshops. 2017.

[16] Yamanaka, Jin, Shigesumi Kuwashima, and Takio Kurita. "Fast and accurate image super resolution by deep CNN with skip connection and network in network." International Conference on Neural Information Processing. Springer, Cham, 2017.

Codebase Reference

- Github codebases: soapisnotfat/super-resolution, SRCNN

Computing Device Description

NVIDIA RTX A5000 (single GPU with 24GB RAM), NVIDIA RTX 2060ti (single GPU with 6GB RAM).

Contributions

Lujia Zhong mainly contributes on the part of Model Selection and Result Analysis. Jingxiang Zhang mainly contributes on the part of Extention and Conclusion. Both two members discuss and evenly contribute to the overall ideas of this project and write the report.