Report on the Image Super-Resolution Project

Xinyue Zhang, Yuebo Yao, Taha Mousavi

1. Introduction

Image super resolution is a process of generating high resolution images from low resolution images [1]. Nowadays, it is heavily applied in medical imaging, surveillance systems and satellite imaging systems [1]. Algorithms for the image super resolution have their own pros and cons and therefore, it is important to understand the mechanisms behind them and the effects created by them. Besides, it is crucial to understand under which conditions the algorithms for image super resolution can be helpful and feasible.

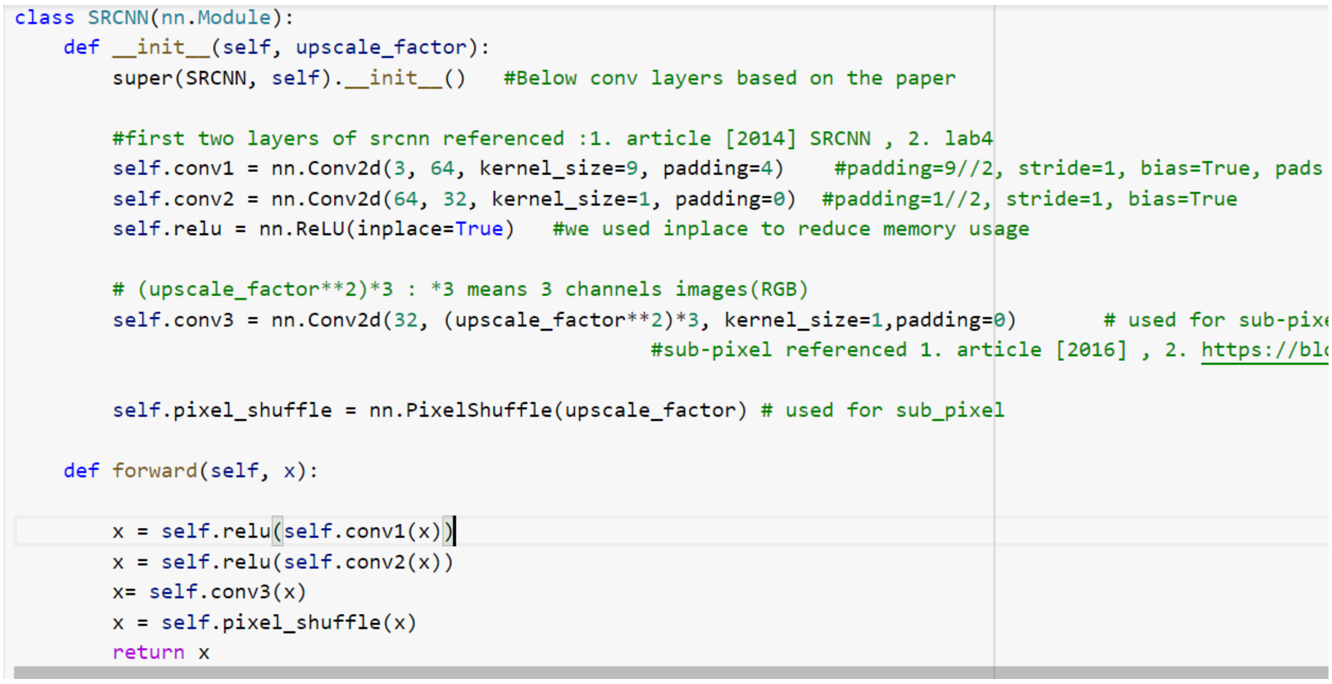
Therefore, our team decided to conduct two comparison experiments to observe the effects of the SRCNN (Super-Resolution Convolutional Neural Network) [2] integrated with sub-pixel convolutions [3], and of the plain bicubic interpolation and discuss their characteristics.

In our study we used the LR_bicubic/X4 dataset and DIV2K_train_HR dataset from DIVERSE 2K resolution high quality images [4], implemented the model by using PyTorch library on the CoLab environment,

2. Description of Algorithm

In the two experiments we did, the algorithm is mainly needed in the first experiment, which is a SRCNN model combined with sub-pixel convolution. While in the second experiment we just apply plain bicubic interpolation to upscale an LR image by 4 times

Hence, we will describe the SRCNN model (by referencing [2]) with sub-pixel convolution for upscaling (by referencing [3]) here. The code of the entire model is as below:



```
class SRCNN(nn.Module):
    def __init__(self, upscale_factor):
        super(SRCNN, self).__init__()   #Below conv layers based on the paper

        #first two layers of srcnn referenced :1. article [2014] SRCNN , 2. lab4
        self.conv1 = nn.Conv2d(3, 64, kernel_size=9, padding=4)    #padding=9//2, stride=1, bias=True, pads
        self.conv2 = nn.Conv2d(64, 32, kernel_size=1, padding=0)  #padding=1//2, stride=1, bias=True
        self.relu = nn.ReLU(inplace=True)    #we used inplace to reduce memory usage

        # (upscale_factor**2)*3 : *3 means 3 channels images(RGB)
        self.conv3 = nn.Conv2d(32, (upscale_factor**2)*3, kernel_size=1,padding=0)       # used for sub-pixe
                                                    #sub-pixel referenced 1. article [2016] , 2. https://blo

        self.pixel_shuffle = nn.PixelShuffle(upscale_factor) # used for sub_pixel

    def forward(self, x):

        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x= self.conv3(x)
        x = self.pixel_shuffle(x)
        return x
```

The SRCNN model we established here is similar as in the paper [2], which uses three convolutional layers respectively for patch extraction and representation, non-linear mapping, and final reconstruction [2].

Specifically, the first layer is used to extract (overlapping) patches from the LR image and represent each patch as a higher-dimensional vector, the vectors include feature maps as many as the vector's dimension number [2] [5]. It takes one 3-channel image as the input and uses 64 filters, the kernel is with size of 9*9, the padding is as 4(which obtained by 9//2) and the stride is 1, a RELU activation function follows the first layer. Through the first layer there will produce 64 feature maps.

The second layer is used to nonlinearly map each higher-dimensional vector produced after first layer to another higher-dimensional vector, which includes another set of feature maps. It takes the 64 feature maps as input, uses 32 filters, the kernel size set as 1*1, the padding as 0(which obtained by 1//2), also the stride as 1. Same as behind the first layer, here will also be followed by a RELU function. Through the second layer 32 feature maps of HR image in the future will be produced.

Now for the third layer, we removed the original one in paper [2], which was used to reconstruct the HR image using the 32 feature maps. Here we establish another convolutional layer combined with sub-pixel convolution for upscaling and reconstruction, which is referenced by [3], the parameters from left to right respectively are: number of inputted feature maps (32), upscaling factor as 4 applied on the 3-channel image, kernel size as 1*1 and padding as 0. And a PixelShuffle method followed is used to implement the upscaling.

For the loss function, we adopt the mean square error (MSE) loss (for which we directly use torch.nn.MSELoss in our code), because it is beneficial to obtain high PSNR value. The formula is as below:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}||F(x_i;\ \theta) - y_i||^2,$$

and the PSNR computing formula is as:

$$PSNR = 10 \times log_{10}\frac{MAX_I{}^2}{MSE} \text{ , [6]}$$

here MAX_I is the maximum possible pixel value of the image, when the pixels are represented by 8 bits, it is 255[6]. In our codes, it is 255.

3. Experiments:

We use DIV2K_train_LR_bicubic/X4 as LR image dataset, and DIV2K_train_HR as HR image dataset [4]. The LR images are regarded as the features in our code while the HR images are regarded as corresponding targets.

3.1 Preparations and executions for first experiment:

We established the 4d-tensor to load LR images and HR images each time, the 4 dimensions in the tensor represent respectively number of batches, channels, image height and image width. The number of batches is 100, and we set the batch size as 1 (namely one image per batch) in our code. Thus, there are a total of 100 images in the dataset. The number of channels is 3 since we use RGB images. For the LR images, we resized each of them into size of (339, 510), while for HR images we resized into size of (1356, 2040). The size of HR images is exactly 4 times of the LR ones. We split the 80% of the images (namely 80 images) as training dataset and 20% (20 images) for the testing dataset.

When training model we use torch.utils.data.DataLoader to load the train dataset, which includes LR images as x_train set and HR images as y_train set in the code. Through training the model, the images in x_train dataset can be reconstructed into new HR images with 4-times upscaled size (namely the predictions), which have the same size as y_train's. When testing the model, we applied the model on the validation dataset, which also include LR images as x_val set and HR images as y_val , the PSNR values between the prediction of x_val and y_val need to be calculated. Here, we also computed the average PSNR value under the current epoch to find the best epoch. Then we plotted the PSNR values for the validation set under this epoch. Then we adopted Adam as the optimizer with a learning rate of 0.001 and trained the model for 70 epochs.

3.2 Preparations and executions for second experiment:

The second experiment for plain bicubic interpolation is simple, we just applied interpolate function imported from torch.nn.functional with mode as 'bicubic' and scale factor as 4 onto x_val dataset, and computed PSNR values between the interpolation results and target(y_val).

3.3 Results:

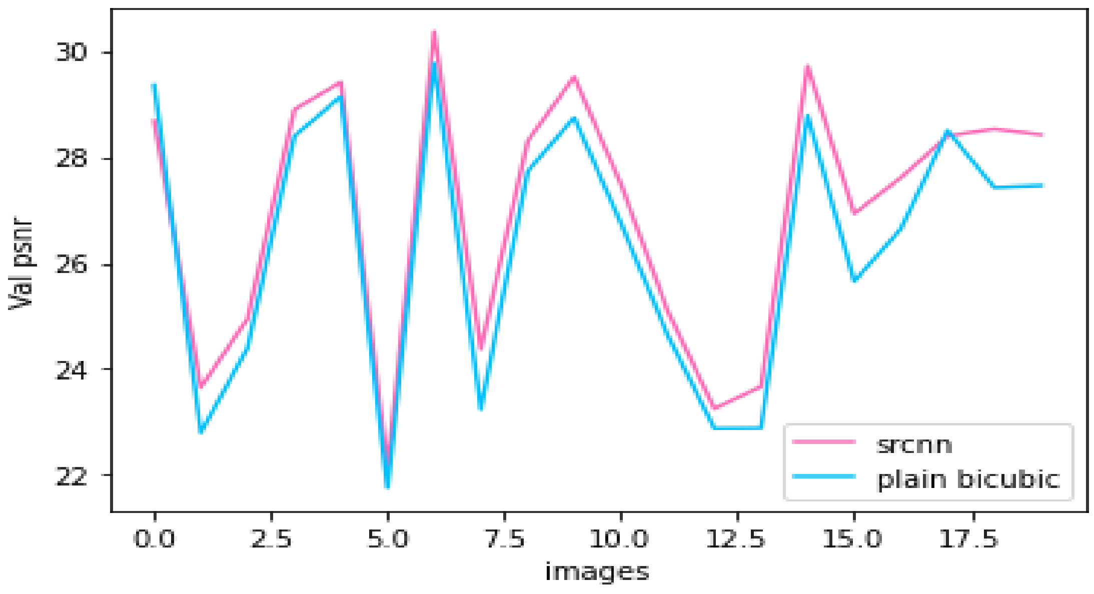


Fig 1. The graph of the Fluctuation of Average PSNR for SRCNN and plain bicubic

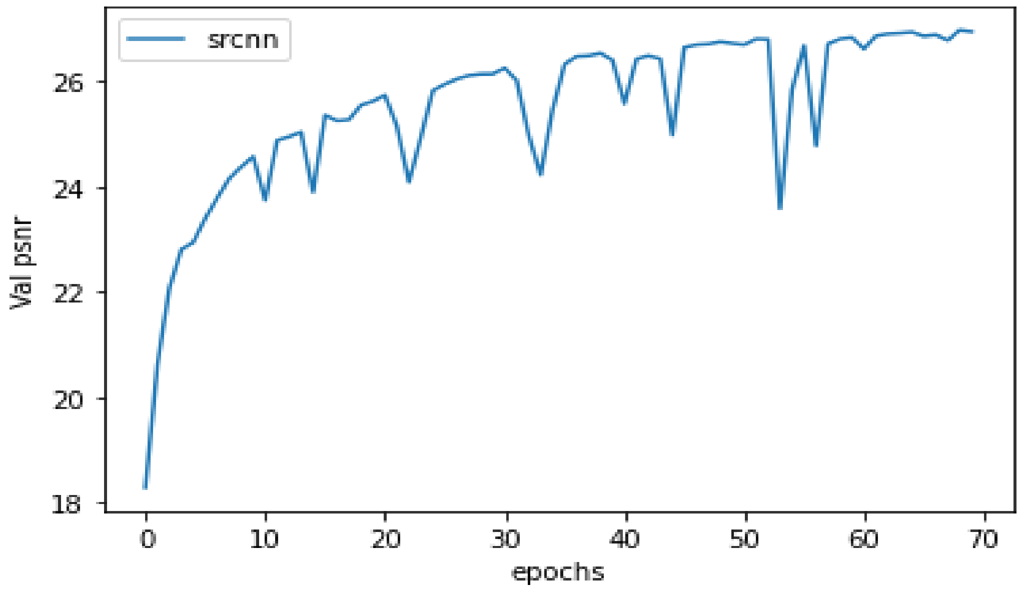


Fig 2. Average PSNR chart for each epoch.

In our experiment, we made the upscaling factor of sub-pixel convolutions as 4 and the same with the bicubic interpolation. Figure 1 shows the fluctuation of the average PSNR along with the number of images processed by the training algorithm with the sub-pixel convolution and that with the bicubic interpolation. As it is shown, the average PSNR fluctuates significantly as the number of images increases. After the 18th image is processed, both algorithms' average PSNR values start to be stable.

The SRCNN model with sub pixel convolutions performs slightly better than the plain bicubic interpolation in terms of the average PSNR value for 20 numbers of images. In our experiment, the average PSNR value for bicubic was **26.34** while for SRCNN it was **26.96.**

The reason for this is the difference between the mechanisms behind two different algorithms. For the sub pixel convolution, the step of upscaling the image is done at the end of the algorithm, and it implies that the image with the smaller size is fed into the neural network and that reduces the memory cost and the computational complexity [8]. Also, an image with a small size can help the convolutional neural network to extract feature maps from the image. However, on the other side, the bicubic interpolation does the opposite and it starts to upscale the image at the beginning of the algorithm [7]. Upscaling means adding more pixels to preserve the actual details of the image in the bicubic interpolation according to [8]. The more upscaling is done, the more details need to be created [8]. Not every piece of detail can be preserved by the algorithm [8]. When to upscale the image creates a difference of results between the sub pixel convolution and the bicubic interpolation.

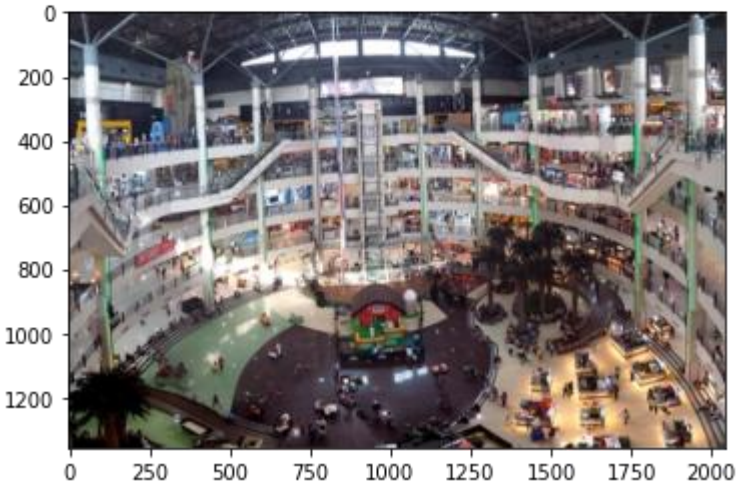


Fig 3. The image processed by the sub pixel convolution

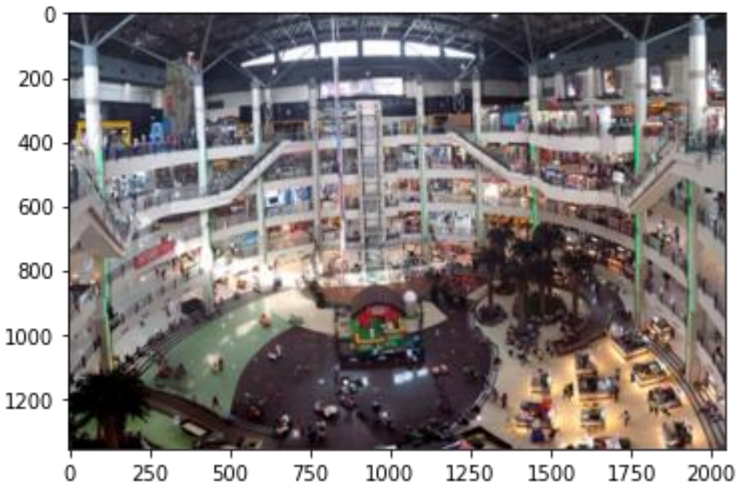


Fig 4. The image processed by bicubic interpolation

Apart from this, interestingly, the difference of the average PSNR between the bicubic interpolation and the SRCNN with sub pixel convolution is not so far away from each other in both models according to Fig 1. This can be illustrated by the difference between Fig 3 and Fig 4. The reason behind it is that the upscaling factor is not that huge. This means for the interpolation algorithm, that there is not so much detail to be filled up. If there is too much detail that needs to be filled, there may be problems like overshooting, clipping, and ringing artifacts as mentioned in the research of [8]. However, a low upscaling factor does not make that happen and that is why Fig 3 looks smooth. For the sub pixel convolution, the step of pixel shuffling is at the end of the procedure according to [7] and because feature maps are extracted before the pixel shuffling, no artificial detail is added to the original image. This avoids problems like overshooting, ringing artifacts. That also makes the output image look smooth. Finally, it can be shown that the quality of the image is not affected by the upscaling factor.

4. Conclusion

Overall, there are pros and cons for using either the plain bicubic interpolation or the SRCNN with sub pixel convolutions. The plain bicubic interpolation only works well when the upscaling factor is not too large, because the algorithm cannot preserve too much detail in the way from the original image. On the other hand, the SRCNN with sub pixel convolutions can work with higher upscaling factors because it only does the pixel shuffling at the end of the procedure and in that way, the detail of the original image is preserved in a bigger image regardless of the output image's size. Still, both algorithms are not perfect and they both have room for some further improvements.

5. References

*The references of codes included in the comments of code.

[1]

• Khandelwal, Y. (2021, May 27). Image Super Resolution | Deep Learning for Image Super Resolution. Analytics Vidhya. Retrieved January 16, 2022, from https://www.analyticsvidhya.com/blog/2021/05/deep-learning-for-image-super-resolution/#free-courses

[2]

• Dong, Chao, Chen Change Loy, Kaiming He, and Xiaoou Tang. "Learning a deep convolutional network for image super-resolution." In European Conference on Computer Vision, pp. 184-199. Springer, Cham, 2014

[3]

• Shi, Wenzhe, 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, pp. 1874-1883. 2016.

[4]

• https://data.vision.ee.ethz.ch/cvl/DIV2K/

[5]

• SRCNN with pytorch implementing Learning a Deep Convolutional Network for Image (2019).

CSDN.
https://blog.csdn.net/xu_fu_yong/article/details/96434132?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522164157138016781683981956%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=164157138016781683981956&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2%7Eall%7Esobaiduend%7Edefault-1-96434132.pc_search_insert_ulrmf&utm_term=srcnn+pytorch&spm=1018.2226.3001.4187

[6]
• Wikipedia contributors. (2021, December 26). *Peak signal-to-noise ratio*. Wikipedia. https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio

[7]
• Cen, Z. (2021, December 14). An Overview of ESPCN: An Efficient Sub-pixel Convolutional Neural Network. Medium. Retrieved January 16, 2022, from https://medium.com/@zhuocen93/an-overview-of-espcn-an-efficient-sub-pixel-convolutional-neural-network-b76d0a6c875e

[8]
• Tabora, V. (2021, December 13). Bicubic Interpolation Techniques For Digital Imaging. Medium. Retrieved January 16, 2022, from https://medium.com/hd-pro/bicubic-interpolation-techniques-for-digital-imaging-7c6d86dc35dc