AI6103 Deep Learning & Applications
Group Project: Blind Super-Resolution With Iterative Kernel Correction

Li Xiaochen, Liu Yaohong, Wang Rui, Wang Yuhang

li0029en@e.ntu.edu.sg, liuy0220@e.ntu.edu.sg, wang1806@e.ntu.edu.sg, ywang145@e.ntu.edu.sg

Abstract

This report is regarding the group project of AI6103 Deep Learning and Application. The objective of this project was to review the paper Blind Super-Resolution with Iterative Kernel Correction and reimplement the algorithms. In this paper, the Iterative Kernel Correction (IKC) architecture was proposed, which is composed of three components - Predictor, SFTMD, and Corrector. In blind SR problems, IKC can produce better results than direct kernel estimation. SFTMD is an effective SR network architecture that uses spatial feature transform (SFT) layers to handle multiple blur kernels. In this report, we will discuss the principle and implementation method of the IKC architecture. We conducted vast experiments to compare the results with the original model. In addition, we try to improve the model based on our reimplemented model, since no open-source code is available. Our project link is **https://github.com/RayW18/AI6103-Deep-Learning-Project**.

Introduction

Single image super-resolution (SISR), which aims to transfer a low-resolution image into a high-resolution image, is a popular topic nowadays. Most of the current SR models assume that the downsampling model is known and can be predefined. In the real world, there is no kernel to use, so blind SR is proposed to solve such a problem. However, these methods are still unable to predict the blur kernel for different images. As a result, noise will have a significant influence on these models.

An Iterative Kernel Correction (IKC) method finds that the outcome is sensitive to kernel mismatch. As a result, it introduces a principle that we can use the predictor to predict the initial kernel. Then, we can use the corrector to correct the kernel by comparing the output image with the input. During this process, the SR image will gradually approach ground truth, as shown in Figure 1. What's more, taking the concatenation of images and blur kernel as input, Spatial Feature Transform with Multiple Blur Kernels (SFTMD) is proposed. These two parts are combined to form the final model.

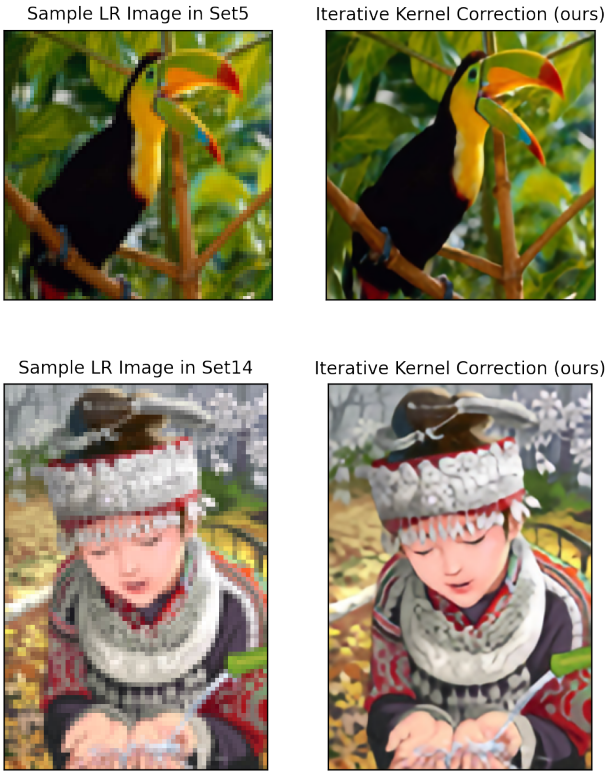

Figure 1: LR image after IKC

Method

In most modern blind super-resolution tasks, the main focus is on the working principles of the image degeneration model, as well as using **learning-based methods** to construct a super-resolver based on the available low-resolution and high-resolution image pairs as training data. In this part, we will introduce the basic mathematical model for this paper.

Degeneration Model Formulation

Degeneration model($\boldsymbol{D}$), the basic mathematic model to explain the relationship between the HR and LR images,

mainly contains the blurry and downsampling process as below:

$$I^{LR} = \boldsymbol{D}(I^{HR}; k, \downarrow_s) + n \tag{1}$$

where we use k to represent the kernel we used for the image blurry; $\downarrow\ s$ stands for the downsampling step and n means the possible noise that occurred in the degeneration process.

In this paper, the authors' team has chosen the most widely used isotropic Gaussian kernel, Gaussian noise, and bicubic downsampling [1] to rebuild the degeneration model as follows:

$$I^{LR} = (k \otimes I^{HR}) \downarrow_s + n \tag{2}$$

Where $\otimes$ means the convolution operation, the corresponding implementation of the blurry step is based on the convolution of the blurry kernel and the original HR images.

Actually, we have a rough understanding of how this degeneration works. Nonetheless, in the real world, the corresponding LR images are much more complex than we have described in the paper: LR images will be degraded by motion blur, the corresponding Gaussian noise will not handle the actual noise type, and so on. We will discuss these challenges in the improvements section.

Recovery Model Formulation

From our point of view, we would prefer to describe the HR images build process as a decoding process: we have got some "enciphered" information (LR images) from the encoder machine (Degeneration model) and try to build a decoder machine to recover the original messages (HR images) from the incomplete messages (LR images). So, the corresponding decoding model mainly contains two parts: find the correct encryption codes (Blurry Kernel) and derive the mapping functions to rebuild the messages (HR images).

Fuzzy Up Sampling Model First let us focus on the most prominent SR model, $\boldsymbol{F}$. Assume we know the exact blurry kernel information used in the degeneration process. What the model $\boldsymbol{F}$ will do is figure out the mapping function between the ground truth HR images corresponding to the input LR image and kernel k. What we want is after the training process, the outputs $\boldsymbol{F}(I^{LR}, k)$ to be similar to the GT HR images.

Kernel Generation Model As a **Blind Super-Resolution** task which means we have no prior knowledge of the blurry kernel information, we need to derive the appropriate kernel to help the SR model $\boldsymbol{F}$ to produce the **perfect visual effect** images which is the main innovation point in this paper. Similar to previous work, the author's team tried to build a predictor $\boldsymbol{P}$ to generate the blurry kernel (from the LR images). And this predictor can be optimized by minimizing the l_2 distance between the generated kernel and the ground truth kernel, as below [1]:

$$\theta_\rho = argmin_{\theta_\rho} \parallel k - \theta(I^{LR}; \theta_\rho) \parallel_2^2 \tag{3}$$

Kernel Corrector Model But is this mathematical most similar kernel shown in the equation above, behaving or in other words can blur the HR image as the ground truth kernel? The answer is probably not. But there may exist many kernels with different widths which have a small distance from the real one: if we choose the distance minimum kernel but with large different kernel widths, the corresponding SR visual results will be totally terrible. Figure 2 [1] demonstrate the sensibility. Due to this mismatch in kernel width, over-smoothing and over-enhancing will occur, greatly affecting SR visual performances and PSNR indicators.

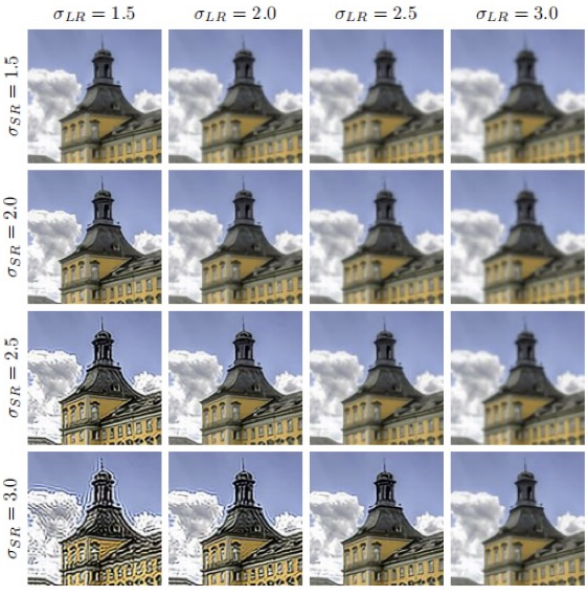

Figure 2: SR sensitivity to the kernel mismatch

To ensure the final performance of SR visual effects, the authors provided another corrector $\boldsymbol{C}$ module to rectify the initial kernel generated by the predictor. The corrector will take the recovered SR results from the initially predicted kernel as input and output the error term k' to rectify the initial kernel and then minimize the l_2 distance from the ground truth kernel as below [1]:

$$\theta_{\mathscr{C}} = argmin_{\theta_{\mathscr{C}}} \parallel k - (\mathscr{C}(I^{SR}; \theta_{\mathscr{C}}) + k' \parallel_2^2 \tag{4}$$

After several iterations of correction, the final estimated kernel will be a balance between similarity to the ground truth kernel and final SR visual effects.

Overall Framework Structure and Working Principles

As we have discussed in previous sessions, there are three key blocks in our model: the SR network $\boldsymbol{F}$, the kernel predictor $\boldsymbol{P}$ and corrector $\boldsymbol{C}$.

Suppose the LR image, I^{LR},we want to recover has the size as $\boldsymbol{C} \times \boldsymbol{H} \times \boldsymbol{W}$, where $\boldsymbol{C}$ denotes the image channels and $\boldsymbol{H}$ and $\boldsymbol{W}$ represent the image resolution. At the start of the recovery work, the predictor will first generate the blurry information according to I^{LR} regarded as h_0. The reason we use h_0 to denote is the space of the original kernel is much larger. We will use the principal component analysis (PCA) method to reduce the computation cost and improve the model generalization ability.

If we denote the original blurry kernel with size $l \times l$, its kernel space dimension is l^2 linear space which is too large to calculate in the steps afterwards. So, this paper mainly uses the dimension reduction matrix $\boldsymbol{M} \in \mathbb{R}^{b \times l^2}$ to project the original kernel space into a much lower dimensional space (equals to b) to make the computation more efficient: h = $\boldsymbol{M}k$, $h \in \mathbb{R}^b$.

Then after we generated the initially projected kernel h_0, we will first test its performance and get the corresponding initial SR images with the $\boldsymbol{F}$: $I_0^{SR} = \mathscr{F}(I^{LR}, h_0)$. And next are the stages for corrector, the pre-trained $\boldsymbol{C}$ will use totally i iterations to update the initial prediction kernel. If we use $\triangle h_i$ to represent the update term for the previously estimated kernel in i^{th} iteration, the corresponding updated estimation kernel results and SR model result using the updated estimations could be calculated as below [1]:

$$\begin{aligned} \triangle h_i &= \mathscr{C}(I_i^{SR}, h_{i-1}) \\ h_i &= h_{i-1} + \triangle h_i \\ I_i^{SR} &= \mathscr{F}(I_{LR}, h_i) \end{aligned} \tag{5}$$

In other words, as the recovery work begins, $\boldsymbol{P}$ will first generate different blurry information according to different images input. Then after several iterations of kernel correctness in $\boldsymbol{C}$, we will get reasonable kernel information for the final recovery to $\boldsymbol{F}$ and generate HR images with good visual effect. The corresponding pseudo-code is shown as Algorithm 1 [1] below:

Algorithm 1: Iterative Kernel Correction

Require: the LR image I^{LR}
Require: the max iteration number t
  $h_0 \leftarrow \mathscr{P}(I^{LR}$(Initialize the kernel estimation)
  $I_0^{SR} \leftarrow \mathscr{F}(I^{LR}, h_0)$(The initial SR result)
  $i \leftarrow 0$(Initialize counter)
  while $i < t$ **do**
    $i \leftarrow i + 1$
    $\triangle h_i \leftarrow \mathscr{C}(I_{i-1}^{SR}, h_{i-1})$(Estimate the kernel error using the intermediate SR results)
    $h_i \leftarrow h_{i-1} + \triangle h_i$(Update kernel estimation)
    $I_i^{SR} \leftarrow \mathscr{F}(I^{LR}, h_i)$(Update the SR result)
  end while
  return I_t^{SR}(Output the final SR result)

Architecture

Network Architecture of SFTMD In this experiment, we combine the **SRMD** [2] stretching strategy with **SFT** layer [3] affine transformation. **SRMD** is a suitable framework to deal with non-image input processing for CNN by using a stretching strategy. Suppose the size of the input blur kernel is $p \times q$, and the noise level is σ. **SRMD** firstly vectorizes the blur kernel and projected onto low dimension by PCA. By concatenated noise level, the dimension of vector is $t+1$. Moreover, the vector is stretched to the size of $W \times H \times (t+1)$ kernel map $\mathscr{H}$. each ith elements of input equals the $\mathscr{H}_i$. Therefore, kernel map $\mathscr{H}$ can be concatenated with the size of $W \times H \times C$ LR image (Low Resolution image). Now, the size of input of CNN turns to $W \times H \times (C+t+1)$. The architecture of CNN is simplified as "3x3 Conv + Pixel Shuffle + Leaky ReLU". However, kernel map dose not contains any image information which may cause result to be influenced by those unrelated interference. This concatenated stretching method is applied on first layer of network, with network going to deeper layer, it has minor influence on whole network. A new SR model is proposed to solve the above problems by adding spatial feature transform (SFT) layers as middle layer named as **SFTMD**. By adding **SFT** layer [3], kernel map $\mathscr{H}$ can influence the outputs via deep network. **SFT** layer learns feature map $\boldsymbol{F}$ from prior on condition kernel map $\mathscr{H}$: $\boldsymbol{SFT}(\boldsymbol{F}, \boldsymbol{H}) = \boldsymbol{F} \otimes \gamma + \beta$. the transformation parameters (γ, β) is donated by CNN with concatenated input $(\boldsymbol{F}, \boldsymbol{H})$. The previous layer feature map $\boldsymbol{F}$ obtain size $W \times H \times C_f$ where $\boldsymbol{C}_f$ is the channel of feature map. At the end, the total size of concatenated feature maps and kernel maps is $W \times H \times (C_f + t + 1)$. To handling multiple concatenated kernels, **SRResNet** [4] architecture is introduced. 16 residual blocks are applied in the network which shares feature map and transformation parameters (γ, β) with **SFT** layer. However, the application of layer is different with the application in sematic SR. In **SFTMD** network, each residual block and skip connection are following a **SFT** layer, which means the network only migrate the transformation characteristic of **SFT** layer and keeps the code map spatial uniform at same time. The architecture of **SFTMD** network is shown on Figure 4.

Network Architecture of Predictor P The architecture of predictor network is shown on Figure 5. For Predictor, we first let the low-resolution image go through four convolution layers with Leaky ReLU activations to generate the kernel spatially and form the estimation map. After that a global average pooling layers will generate the estimated kernel map h_0 by taking the mean value spatially. The h_0 is later used in the corrector process.

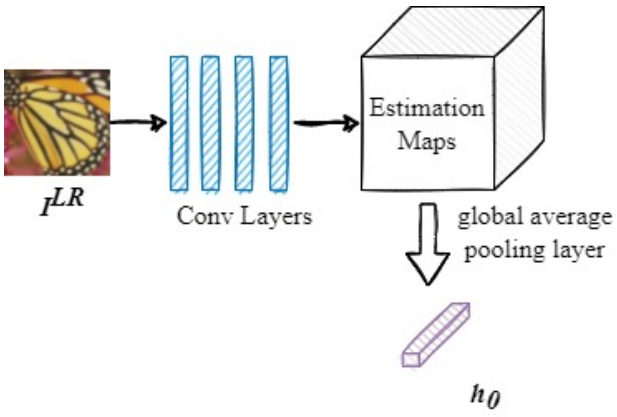

Figure 5: Architecture of Predictor

Network Architecture of Constructor C The architecture of **corrector** network is shown on Figure 6. For the predictor, we firstly use the **SFTMD** model with the input l_{LR}

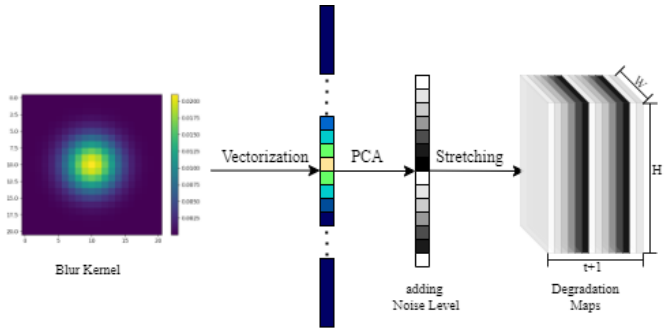

Figure 3: PCA Process

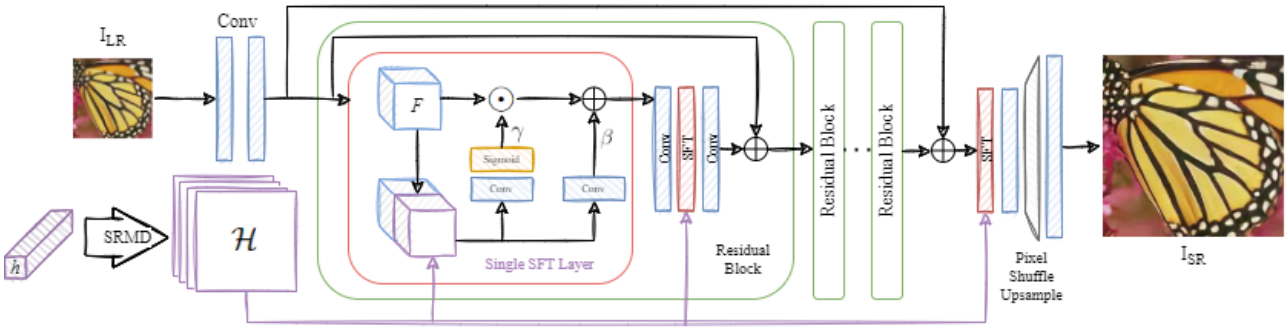

Figure 4: Architecture of SFTMD

and h_0 generated by the **predictor** to generate I_{SR}. After that, we take the I_{SR} and the previous estimation h, which is first output by **C** and later updated by corrector, as inputs. We make several iterations to update the estimation h by using the equation 4. The input I_{SR} firstly goes through seven convolution layers with Leaky ReLU activations to generate feature maps F_{SR}. In the meanwhile, the input estimation h is processed by two fully-connected layers with Leaky ReLU activations in order to extract the internal correlation of the previous kernel, which is f_h. After that, we then use the same method used in SFTMD to stretch the f_h into F_h, concatenated to F_SR to predict $\triangle h$. In this step, we use three convolution layers which has the**1 × 1** kernel size, and Leaky ReLU activations to estimate for $\triangle h$ spatially. After that, we use a similar way in ***P*** to compute the global estimation of $\triangle h$ with a global average pooling operation. Estimate

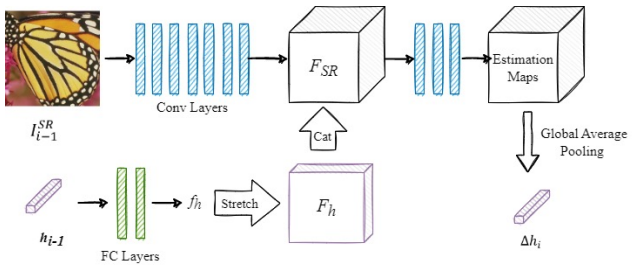

Figure 6: Architecture of Corrector

Experiments

Dataset Preparation

Actually we initially plan to use the complete 2K 3450 images including DIV2K and Flickr2K to train different kernel width ranges for different recovery size. But considering about the GPU hash rate we have, we only use the corresponding 3341 256*256 images in Flickr2K dataset instead to just train the network recover the initial RGB image to its 4 times size(SR factor equals to 4). And for the model evaluation part, we mainly use Set 5 and Set 14 to calculate PSNR and SSIM for the recovery images and GT.

Training Procedure

As we have mentioned before, during SFTMD training process, we mainly random select the kernel width in **0.2** to **4** for SR factor 4 to simulate the blurry process on the original HR images. Then after a bicubic down sampling process, we will get the low resolution, blurry images. Finally, we will use the corresponding selected kernel and LR images as SFTMD inputs and use **l1 loss** function for the recovery images from the SFTMD with GT to perform the backward propagation. Then for the predictor and corrector, they could be alternatively trained by **l2 loss** function when we provided a pretrained SFTMD model. The corresponding working iterations we set is just the same as the paper set,7, which is enough to rectify the predictor outputs. Finally, we use the Adam optimizer with learning rate equals to **1e-4**, beta 1 equals to **0.9**, and beta 2 equals to **0.99** for optimization.

We mainly use the Pytorch Framework on colab with A100-SXM4-40GB to train our models.

Results

We mainly evaluate IKC model on both synthetic Test Images and real image set. For synthetic images set, we mainly calculate the output HR images PSNR and SSIM with GT for different model components. After that we will use the real world images to specifically see the visual recovery effects.

1. On synthetic image sets

(1) SFTMD Model Evaluation

The specific PSNR for SSIM values using our model recovery is shown as the row 4-5 of the table 1.

We can see that the model we rebuild has a approximately performance on PSNR and SSIM compared to the paper value. For the same number of images in training set, we can see that the final PSNR and SSIM when we use the corresponding high resolution (512*512) images to train is higher than using low resolution (256*256) images in training process. And it could be the main reason for our rebuild model has a little bit lower PSNR and SSIM than the paper.

(2) IKC Model Evaluation

In this part, we will first focus on the performance on individual predictor and corrector performance and then step into the complete IKC evaluation stage. Similarly we will also use PSNR and SSIM as our evaluation indicators.

- Individual component test

  For predictor test part, we directly use the kernel predicted from P to recover the LR image in F and calculate the difference with GT.

  And for corrector, to better test its correction capability, we only provide a not well-trained predictor to provide the initial estimated kernel information and see whether the corrector could rectify the kernel map in several small iterations. We can see that for the sample images in Set 5 and Set 6 shown in figure 7 below, the initial kernel estimation results is not perfect: with **22.384dB** and **18.577dB** on PSNR respectively , but after a few correction iterations, the final PSNR increased to stable **25.973 dB** and **20.379dB**. At the same time the corresponding details of the image could be more clear.

- Combination Test

  And finally for the complete IKC model testing, we will use a well-trained SFTMD Predictor and Corrector to do super-resolution task. The results are shown in the eighth and ninth row in table 1, the final PSNR and SSIM on both sets are also similar to the paper said. Although our reimplement IKC has a bit lower PSNR, it has similar SSIM on set 5 and higher SSIM on set 14 which indicates our model could better maintain the structure property.

2. On Real Image Set

We have randomly selected some low-resolution blurry images within kernel width in our model range on the Internet. Because we do not know any blurry kernel prior information, this single super-resolution task is blind. Since there does not exist their ground truth images for PNSR and SSIM calculation, we only focus on their real recovery visual effect. Just as the paper presented, we mainly focus on the images detail's part, for example the blurred building roof and letter's pixels in the sample images in figure 8.From the result images, we can see that the area blurred in the original figures have been repaired to be clearer. The IKC model could output a pleasuring SR results.

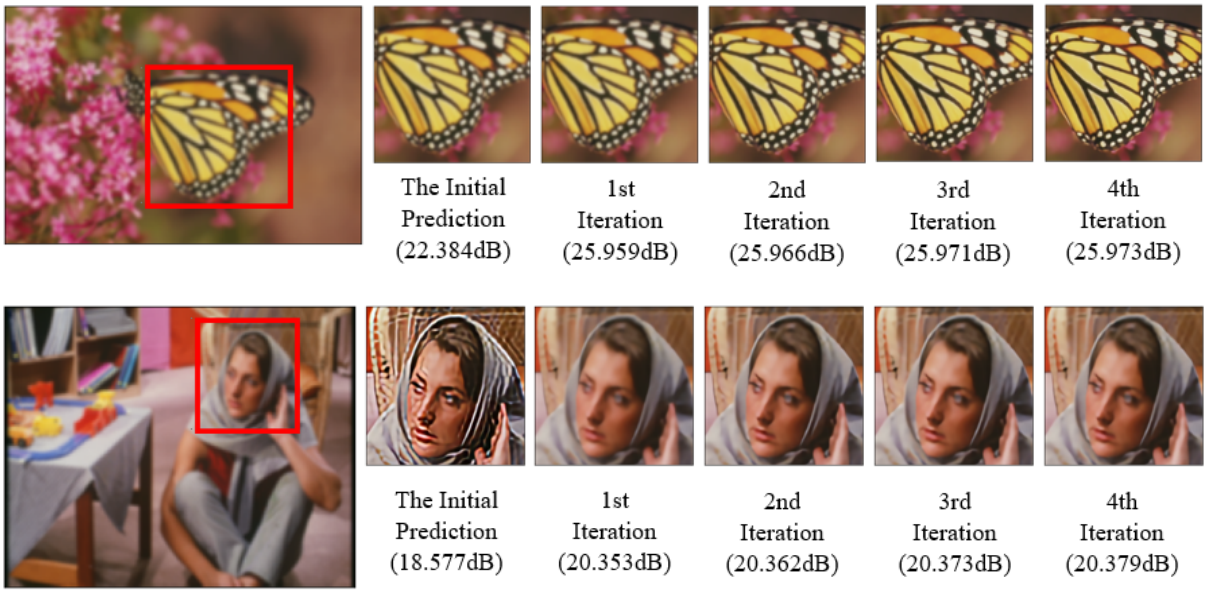

Figure 7: The intermediate SR results during the process of corrector

model	set5		set14	
	PSNR	SSIM	PSNR	SSIM
SFTMD (Paper)	32.05	-	28.55	-
SFTMD (Ours - 256)	27.46	0.8876	24.90	0.8161
SFTMD (Ours - 512)	29.03	0.9110	25.80	0.8301
P + SFTMD (Paper)	29.29	0.9014	26.40	0.7137
P + SFTMD (Ours - 256)	27.32	0.8923	24.37	0.8111
P + SFTMD (Ours - 512)	27.90	0.9100	24.44	0.8220
IKC (Paper)	**31.52**	**0.9278**	**28.26**	**0.7688**
IKC (Ours - 256)	28.24	0.9020	25.32	0.8242
IKC (Ours - 512)	**29.43**	**0.9229**	**26.05**	**0.8412**

Table 1: test results on PSNR and SSIM

LR Images IKC (Ours)

Figure 8: SR results in real image

Improvements

From the paper, authors consider the noise is following Gaussian distribution for the whole image. However, we realized the assumption will create high weight of noise on the center part of image. In real life, the noise is random and can be not predictable. Therefore, to simulate the real-world noise, we proposed a random noise model which is not just have high chance appear on the center of picture but also have even chance to appear on the image corners.

To make this proposal to be true, we divided the HR_image to multiple 8*8 small segments, each of small segment follows Gaussian distribution which means the pixel noise is similar to its surroundings pixels. At same time, we created a random number (0-75) of the noise sigma, when sigma equals to 0, means no noise on this segment, when sigma equals to 75, means there is high distribution Gaussian noise on this segment. By using this way to simulate the real-word noise, our model has the better training strategy comparing with just a single distribution for the whole image.

Conclusions

During this paper rebuild project, we have explored how the kernel width mismatch problem will affect the final super resolution results and successfully get a satisfactory model performance. We tried to build a more complex noise model to adapt to the various real word image, but its performance has not been increased apparently. Just as the paper guided, we may try anisotropy blur kernels to fix the images under motion blur problem in the future.

References

[1] Gu, Jinjin and Lu, Hannan and Zuo, Wangmeng and Dong, Chao. Blind Super-Resolution With Iterative Kernel Correction. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 1604-1613

[2] Kai Zhang, Wangmeng Zuo, and Lei Zhang. Learning a single convolutional super-resolution network for multiple degradations. In IEEE Conference on Computer Vision and Pattern Recognition, volume 6, 2018. 1, 2, 3, 4, 6, 8

[3] Xintao Wang, Ke Yu, Chao Dong, and Chen Change Loy. Recovering realistic texture in image super-resolution by deep spatial feature transform. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 606–615, 2018. 2, 4, 5

[4] Xintao Wang, Ke Yu, Chao Dong, and Chen Change Loy. Recovering realistic texture in image super-

resolution by deep spatial feature transform. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 606–615, 2018. 2, 4, 5