Dingyan Chen

ID Number: 4276853

Supervisor: Bai Li

Module Code: G53IDS

2018/04

Image Super-resolution for OCT and CT images using deep learning method

Submitted Apr.2018, in partial fulfilment of
the conditions for the award of the degree **BSc Computer Science and Artificial Intelligence.**

Dingyan Chen
4276853
School of Computer Science
University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature _____

Date _____/_____/_____

1 Abstract

In this dissertation, I will present a method for the image to improve its resolution. The process of transfer a low-resolution image to high-resolution one is called image super-resolution. OCT and CT images are a type of dedicated medical image, which is of three-dimensional. I will design a method for image super-resolution on OCT and CT images, using the state-of-the-art methodology called generative adversarial network. Followed by the implementation of SRGAN, I will propose a new evaluation criterion and try to maximize the performance of super-resolution on those type of images. Nevertheless, challenges such as long-time training, tune parameter settings and loss function are raised throughout my project. Solving most of the challenges, I still explore more issues and propose new questions and expectations for the future work.

2 Acknowledgements

I would like to express my sincere thanks and gratitude to those people who helped me during the whole project, especially my supervisor Bai Li for her unwavering support, guidance, encouragements and patient solution to my questions.

I also would like to extend my thanks for those people who finish my questionnaire in the final stage of project. Without the support of them, I can't develop the evaluation criterion with actual and precise data in this project.

Table of Contents

1 Abstract 2

2 Introduction 6

3 Motivation 7

4 Related Work (Literature Review) 8

4.1 Traditional Method (SR based on reconstruction) 8

4.2 Sparse-coding-based method (SR based on learning) 10

4.3 Image super-resolution using deep convolutional neural networks (SRCNN) 10

4.4 Description of performance and comparison 11

4.5 Reflection and consideration after the literature review 13

5 Description of the work 13

6 Methodology 13

7 Background 14

7.1 Generative Adversarial Network (GAN) 14

7.2 ResNet 16

7.3 How GAN and ResNet contribute to image super-resolution 17

8 Design 18

8.1 How to get perceptually good result 18

8.2 Adversarial Network Formula 19

8.3 Network Structure 19

8.4 Perceptual Loss Function 21

8.4.1 Content Loss 21

8.4.2 Adversarial Loss 21

9 Implementation 22

9.1 Pre-processing 3D OCT and CT image 22

9.2 Implementation of Algorithm 23

9.2.1 Platform Chosen 23

9.2.2 Language and Framework Chosen 23

9.2.3 Code Design and Writing 24

9.3 Train the model 29

9.4 Challenges and problem encountered 31

10 Evaluation 32

10.1 Evaluation Criterion 32

10.2 Testing process 35

10.3 A view of result 35

10.3.1 OCT image 35

10.3.2 CT Image 36

11 Summary and Reflection 37
11.1 Project management covering and challenges encountered related to project 37
11.2 Contributions and Reflections 38
11.3 Future Work 39
12 Bibliography 40

3 Introduction

This project is an applied research project based on experiment and exploration. It contains numerous literature review in the early phase, combining the state-of-art achievements with personal knowledge, to develop an application solving the real-world problem.

In our daily life, we always come up with the following situation: Once you go back from a trip and you get a series of fabulous pictures, but some of them might be blurred due to uncertain reasons. You like these images and want to improve their resolution. Therefore, having a technique that can improve the image resolution is quite meaningful and practical. Actually, image super-resolution is a classical problem in computer vision field(Irani,1991).

Super-resolution(SR) is the process to transform low-resolution(LR) image or video to high-resolution(HR). The potential impacts of image SR range from medical, astronomy and geology. As shown in Figure 1, it overcomes the limitation of our surveillance cameras, enhancing the resolution and diagnose to medical image and satellite image.

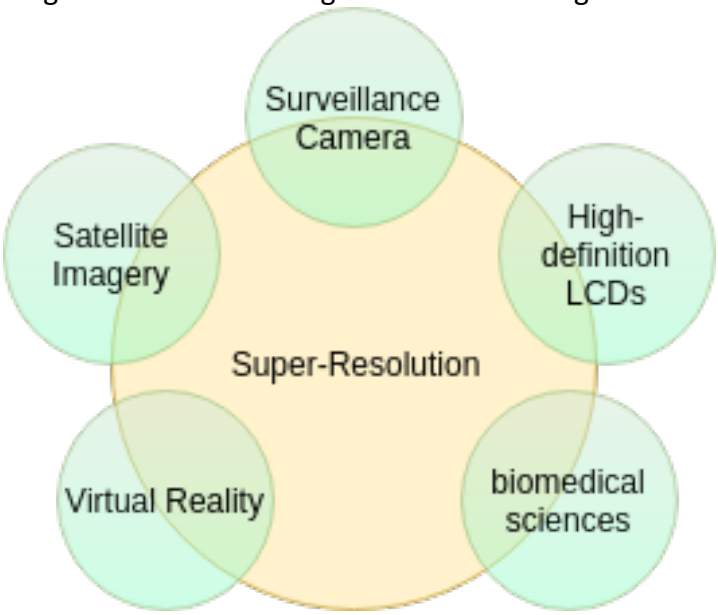


Figure 1: The different usage of super-resolution in real life

There are loads of research and studies before, but most of them are traditional methods like sparse-coding-based method and Bicubic method. Recent state-of-the-art methods for single image super-resolution are mostly related to machine learning, especially deep learning. The most famous one in this field is called super-resolution using a deep convolutional neural network(SRCNN). It starts people to think about using the neural network to improve the resolution of the image. The typical idea is to use abundant low-resolution images to learn and create a high-resolution result. More detailed explanation and analyze will be elaborate in this dissertation. However, Ian Goodfellow proposed a brand new deep learning framework in 2016 called Generative Adversarial Nets(GAN), which provided an innovative platform for people to optimize image super-resolution.

In this paper, I will first analyze the advantages and drawbacks of GAN and explain how it combined with Residual Networks(ResNet) and be applied in image super-resolution for 3D

Computed tomography(CT) image and optical coherence tomography(OCT) image. It is shown that CT and OCT images are in low resolution and 3-dimensional but will be used frequently in the medical field. Meanwhile, 3D-SRGAN exposes a few issues that will affect the result of super-resolution. I will point out those issues and challenges, analyze and modify them.

The primary aim of this project is to develop an application or algorithm that will help medical workers with their analysis of CT and OCT images by improving the scale and the resolution of the images. Though particularly focused on medical images, the algorithm will also have a positive effect on people's daily life and offer an innovative inspiration for scholars and computer scientists in the future. In this dissertation, I will begin with a literature review explaining past research including traditional methods and deep learning methods. Then I will analyze how GAN can be applied in the same way as CNN did in image super-resolution. Furthermore, there will be an introduction of perceptual loss function in GAN, which is totally different from the previous neural network methods. After that, I will demonstrate the implementation of super-resolution based on GAN and ResNet and how it will be used in the medical image. Afterwards, I will evaluate the algorithm by comparing SRGAN with SRCNN in terms of various criteria, including time cost, PSNR(Peak Signal to Noise Ratio) value, SSIM(structural similarity index) value and MOS(Mean opinion score) value. I will introduce my time management and research deployment in the next section. Finally, there will be a reflection section for me to illustrate my academic and personal contribution to the whole project. The project is concluded with a summary and a brief prospect to future work.

4 Motivation

The motivation of doing a super-resolution on Computed tomography(CT) image and optical coherence tomography(OCT) image is comprehensive. It can be explained through two aspects. First, it contributes to the medical area and computer vision area. It is intuitive that the social environment requires the solution to this pertinent problem. Super-resolution is a theory, but it can be applied in real industry to solve the actual situation and make the society a better and more harmony place. Especially in the medical area, imagine that the doctor scans your brain or eyeball but unable to find the symptoms because of the blurred photograph. Also, the cost of the optical camera can be reduced as well as diagnose fee. OCT is an established medical imaging technique that uses light to capture micrometer-resolution, three-dimensional images from within optical scattering media(e.g. biological tissue)(Hui, 2017). However, OCT images are usually of low resolution. We want to improve the resolution and apply it in the real medical industry field. Developing an end-to-end application will hugely improve the medical performance and reduce time cost in the therapy process.

Furthermore, most state-of-art research is based on 2D single image super-resolution. However, what we did is 3D image super-resolution. 3D OCT image is a stack of 2D images, usually of 30-100 images. It is quite challenging because if one of the results of the 2D image is not satisfying, the whole 3D image will be totally crash. Therefore, it also requires us to find an outstanding method for image super-resolution.

5 Related Work (Literature Review)

This section introduces past research on image super-resolution regarding three aspects: traditional super-resolution based on reconstruction, traditional super-resolution based on learning and modern convolutional neural network method(SRCNN). In each aspect, I will give the process of the method, including some necessary mathematical formula. To better demonstrate the process, there will be clear chart describing it. Followed by another chart and a table, I will give a comparison of both methodology especially on its performance. In the end, I will show you my reflection and consideration after the literature review in the early stage.

5.1 Traditional Method (SR based on reconstruction)

The image super-resolution has been developed decades ago. Briefly, it is an inverse process of the high-resolution image to sequence low-resolution image. As shown in Figure 2 that the HR image can transfer into a sequence of LR images through warping, blurring, decimation and added noise. The process can also be expressed in the mathematical formula:

$$y_{1\ldots M} = D_k B_k W_k x + \boldsymbol{n_k},$$

note that

- x is the high-resolution image
- W is Warping, B is Blurring, D is decimation (one sampling method)
- n is the noise

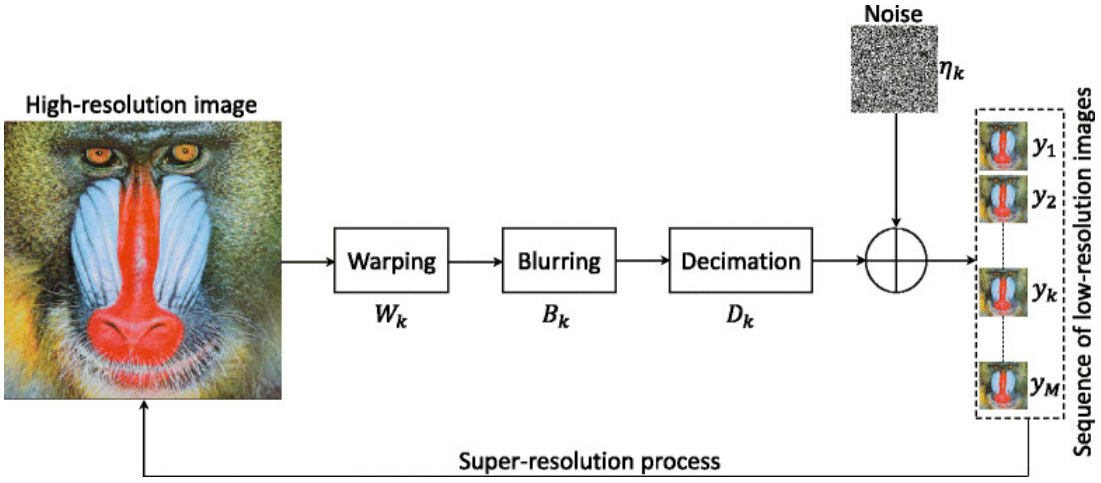


Figure 2: The process and basic principle of HR image to LR image

Therefore, how to change an LR image to HR? The answer is obviously inverse. The following chart is the process:

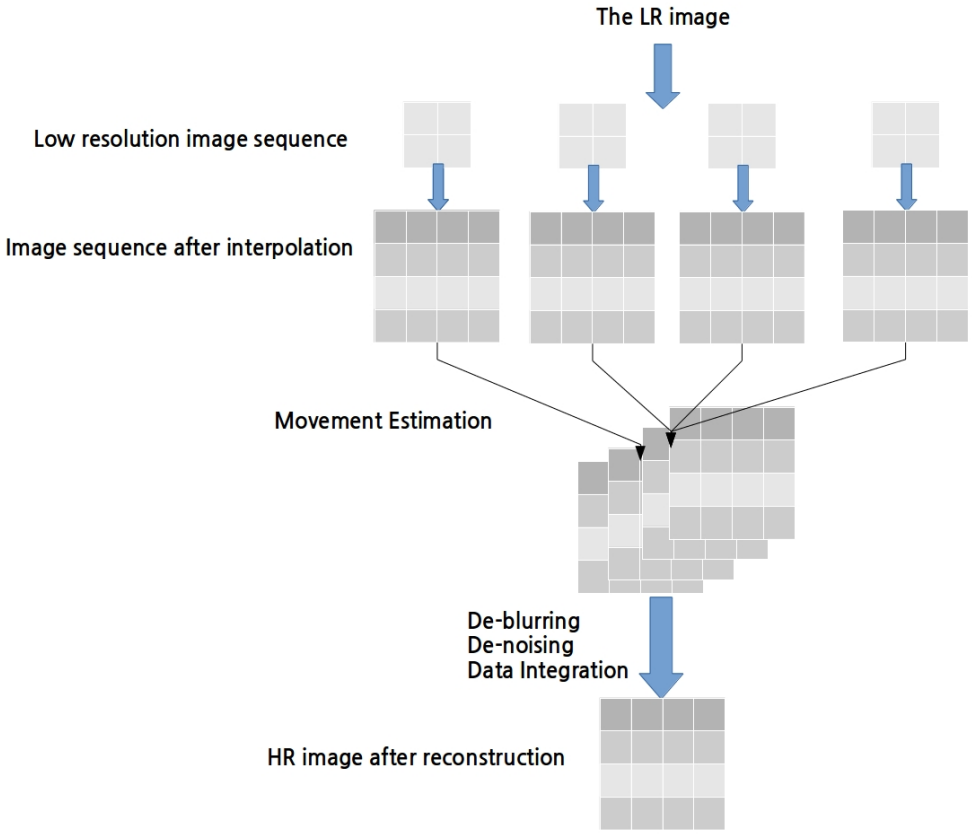


Figure 3: The process of LR image to HR image

The low-resolution image firstly extracts the smaller image piece sequence and then do interpolation, which means insert some similar value into our pixel's. From the figure 3 above, we can see some 2*2-pixel image change to 4*4, and after movement estimation, we integrated those image sequence to a bigger image, which turns out to be our high-resolution result. Figure 4 is the result from Bhatt's paper "Comparative Analysis of Interpolation and Texture Synthesis Method for Enhancing Image", illustrated the result of different interpolation algorithms, from which Bicubic is the best. Also, Bicubic will be used in our SRCNN method afterwards.

INTERPOLATION ALGORITHMS	PSNR(dB)
Nearest Neighbour	26.05
Bilinear	27.12
Bicubic	27.18

Figure 4: The comparison of different interpolation algorithm. PSNR is a criterion checking which algorithm performs better in resolution improving (Bhatt et. al, 2013).

5.2 Sparse-coding-based method (SR based on learning)

The sparse-coding-based method is one of the representative methods for external example-based image super-resolution (Dong et al., 2014). This method firstly extracts overlapping patches from the image and pre-processed (e.g., subtracting mean). These patches are then encoded by a low-resolution dictionary, as shown in "Sparse Coding Stage" in Figure 5 below. The sparse coefficients are passed into a high-resolution dictionary for reconstructing high-resolution patches. Finally, the overlapping reconstructed patches are averaged to output the result (Zhang, 2015). In sparse-coding based SR, it is comprehensive that people tend to focus on learning and optimizing the dictionaries or alternative ways of modelling them. However, this has not been optimized to a higher performance or find a better way to modelling.

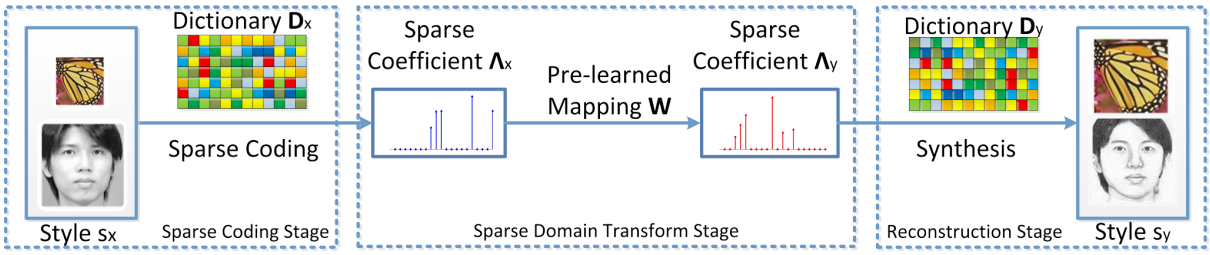


Figure 5: The process of sparse-coding-based method

5.3 Image super-resolution using deep convolutional neural networks (SRCNN)

SRCNN is the first deep learning approach in super-resolution field. Unlike other traditional methods introduced above, it either exploits internal resemblance of the same image, or extracts dictionary function and mapping rules from external low resolution and high-resolution image pairs(Dong et al.,2014). It directly formulates a deep convolutional neural network and learns the mapping of low resolution and high-resolution image. With a delicate neural network structure, it displays an exceeding performance over any other traditional methods. The process of SRCNN can be described as the following 4 stages:

1. Preprocessing
2. Patch extraction and representation
3. Non-linear mapping
4. Reconstruction

Preprocessing: Imagine that we have got a single low-resolution image of any size. What we first need to do is to upscale the image to the desired size using interpolation. From the figure 4 above, bicubic interpolation might be the best choice. Denote the interpolation image as **Y**.

Patch extraction and representation: This operation extracts patches from the low-resolution image Y and represents each patch as a high-dimensional vector (Dong et al., 2014). It is believed that each convolutional layer in CNN will extract a feature map to us from every filter. Therefore, the first layer can be demonstrated as an operation F1:

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

where W_1 and B_1 represents the filters and biases respectively. Here the size of W_1 is $c * f_1 * f_1 * n_1$, where c is the number of channels in the input image (typically it will be 3, which indicates RGB), f_1 is the spatial size of a filter and n_1 is the number of filters. So, in the convolutional layer, each convolution has a kernel size of $c * f_1 * f_1$. The output is composed of n_1 feature maps (Dong et al., 2014). And bias, obviously, will be in size $c * f_1 * f_1$, and there

will be n_1 biases, so B_1 is an n_1- dimensional vector. After the linearly mapping, we use an activation function called ReLU to non-linearly mapping the operation. ReLU is max(0, x).

Non-linear mapping: This operation nonlinearly maps each high-dimensional vector from the above operation to another high-dimensional vector. In CNN, non-linear mapping actually doesn't have any impact on the feature itself. The feature of the image will always be retained. Meanwhile, the non-linear mapping is not just used any activation function here, we nonlinearly operate every element pixel-wise, that is, to have a filter size of 1*1 and stride is 1. The operation of the second layer is:

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

where W_2 is of a size $n_1 * 1 * 1 * n_2$. This is because of that from the last layer, we have an output depth of n_1, and each is $c * f_1 * f_1$ and this time we want to increase it to n_2. Meanwhile, B_2 is n_2 $dimensional$. Each of the output in the second layer is conceptually a representation of a high-resolution patch that will be used for reconstruction (Dong et al., 2014).

Reconstruction: This operation aggregates the above high-resolution patch-wise representation to generate the final high-resolution image. We define the final layer like this:

$$\mathrm{F}(Y) = W_3 * F_2(Y) + B_3.$$

To change the depth of channel back to c, the W_3 is of a size $n_2 * f_3 * f_3 * c$.
A typical setting of these hyper-parameter, from the experiment, is expected to be $f_1 = 9, f_3 = 5, n_1 = 64, n_2 = 32$.
We can visualize our framework as it shown in Figure 6. Then there is only one question remained, what is our loss function?
We should minimize the loss between the reconstructed image F and high-resolution image X, and what we use is the **Mean Squared Error (MSE)**:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n} \left|\left|F(Y_i;\theta) - X_i\right|\right|^2,$$

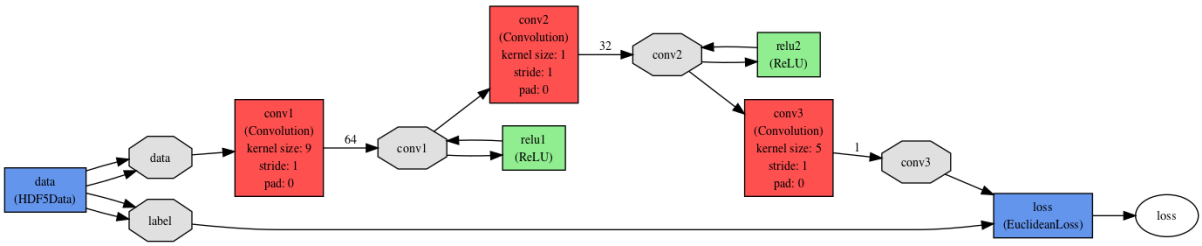


Figure 6: Visualize the CNN network in caffe

5.4 Description of performance and comparison

SRCNN uses deep learning neural network-- Convolutional Neural Networks to extract main features of the image and consider an end-to-end mapping between low-resolution and high-resolution images. In this approach, the entire SR pipeline is fully obtained through learning, with little pre/post-processing.

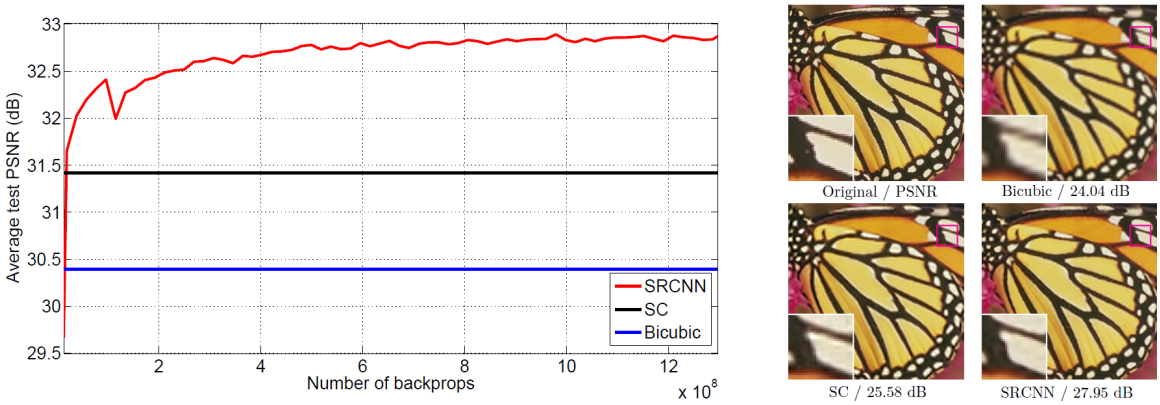


Figure 7: Comparison of Bicubic interpolation, sparse-coding and SRCNN

From the figure 7 above, we can see that as the number of backpropagation increase, SRCNN has higher PSNR value, which explained to be criteria of image resolution above, than bicubic and sparse-coding based method. The performance may be further improved with more training iterations. (Dong et al., 2014)

The following table describes each method's strengths and drawbacks:

Method	Advantages	Disadvantages	Example
SR based on reconstruction	A fully-developed technique and has a great effect in some situation by enhancing the effect and predict movements.	1)It is difficult to always predict movement in a pixel level. 2) Directly use an image, don't use the pre-processing method.	Traditional interpolation SR
SR based on learning	Integrate machine learning and pattern recognition algorithm and technique, reduce computational cost.	1)Need abundant data(example-based) 2)Learning and training process is long	Sparse-coding based SR
SR based on deep learning approach	Use deep learning technique, obtained the best performance among all	Still need to figure out what neural network structure and hyper-parameter is the best. If we use bad parameter, the result is totally crashed.	SR based on deep convolutional neural network

5.5 Reflection and consideration after the literature review

Actually, what SRCNN did was create an innovative platform for computer scientists to consider super-resolution based on deep learning method. It still exposes a huge number of drawbacks. For example, the neural network structure is defective as deep learning developed. Overfitting is generally considered as the most significant issue when people are training their network. There are lots of advanced neural network structure and design like ResNet to solve overfitting in this case but how can we apply them to super-resolution? What I intended to do is to develop a better approach based on other deep convolutional neural network frameworks.

6 Description of the work

In this section, I will analyze what my project is meant to achieve in the end, what is the ultimate goal and what questions my project should solve or optimize.

According to section 4.5, SRCNN raises issues like overfitting, which eventually learns bad parameter after training. Therefore, I designed the following question for me to think about throughout the project.

- How do we avoid overfitting for the neural network?
- How do we evaluate whether the result of the high-resolution image is good or not? The current method calculates PSNR, is it actually the only one and the most precise one?
- How can we develop a method that will solve 3D image super-resolution?
- Obviously, the image obtained by minimizing MSE is too smooth. MSE cannot capture the perceived difference between the model output and the truth image. Imagine a pair of images, the second one copied the first but changed a few pixels. For humans, replicas and originals are almost indistinguishable, but even such subtle changes can make the PSNR significantly lower and MSE higher.
- What pre-processing work should be done for 3D OCT or CT image?
- The traditional method generally deals with a smaller magnification. When the magnification of the image is above 4, it is easy to make the result appear to be too smooth and lack some realism in detail. How to deal with it?

7 Methodology

In this section, I will explain the methodology I used throughout my project. It will be generally divided into three parts, sources of data, the procedure of study and testing result collection. When it refers to the source of data, the project uses in-depth quantitative data analysis (Socscidiss.bham.ac.uk, 2018), the data is collected from the hospital in a rigid way. All the data of OCT or CT image is Digital Imaging and Communications in Medicine(DICOM) format. DICOM format refers to a medical image scanning format for experts in storing and transmitting. Furthermore, it is a secondary analysis project because data is not collected by myself but is already available. The OCT and CT image is collected and transmitted by my supervisor(Bai Li).

In terms of the procedure of study, the methodology is called the pilot case, also known as the demonstrator (Demeyer, 2018). Here is an idea that super-resolution for a single 2D image is valuable. My work is to prove and apply it also works for OCT and CT image. My project takes the strengths and modifies what is inappropriate in this case. I prove super-resolution algorithms work for the OCT and CT image by building a prototype, a program. Meanwhile, I use deductive reasoning and inference, which deduct that GAN can be used in image super-resolution. The premise is that CNN can be used in image super-resolution and GAN is similar to CNN. GAN is recently a famous deep learning approach that people like to use and explore. CNN, similarly, is another technique that people like to explore before GAN. Therefore, according to my deduction, the conclusion should be GAN can be used in image super-resolution because it inherits the same attribute CNN has.

Finally, as for testing result collection. If you have a look at the evaluation chapter, you will find a new evaluation criterion is proposed. This criterion will be related to human interaction by collecting people's evaluation from the questionnaire, which will be used in testing. Therefore, the collection of evaluation uses quantitative data analysis, which turns people's rate into meaningful data for project testing.

8 Background

In this section, I will explain some related background knowledge that I used in my project. By understanding this knowledge, it will be clear that I will introduce what is GAN and ResNet, and why it is appropriate for me to use the methods in my project on an academic basis.

8.1 Generative Adversarial Network (GAN)

Introduced by Ian Goodfellow, Generative Adversarial Network is a class of unsupervised machine learning algorithm that two networks contesting with each other in a zero-sum game framework (Goodfellow et al, 2014). In this chapter, I will give a comprehensive explanation of GAN including the generator network and discriminator network, its loss function and pseudocode. With the help of a detailed understanding of GAN, it will be easier for you to understand the design and implementation in the next chapter.

Generally speaking, there is two networks in GAN, as the name shown "adversarial". One network is generator network, which always generates something to "fool" the discriminator network. One network is a discriminator, which judge the result from generator network. In this way, we can regard discriminator network as the police and regard generator network as the cheater. This process corresponds to a minimax two-player game(Goodfellow et al, 2014), where G model receives a random noise Z, generate an image from the noise, called G(z). Simultaneously, D model is a discriminative network that determines whether a picture is "real." Its input parameters are x, x represents a picture, and output D(x) represents the probability that x is a real picture. If it is 1, it means that 100% is a real picture, and the output is 0, it means that it cannot be true. In the training process, the goal of the G model is to generate real images as much as possible to deceive the D model. As shown in Figure 8, the goal of D is to separate the generated image from real images. In this way, G and D constitute a dynamic "game theory process". What is the final result of the game? In the ideal state, G can generate a picture G(z) that can "cheat" D, and for D, it is difficult to determine whether the generated image is true or not, so D(G(z)) = $\frac{1}{2}$.

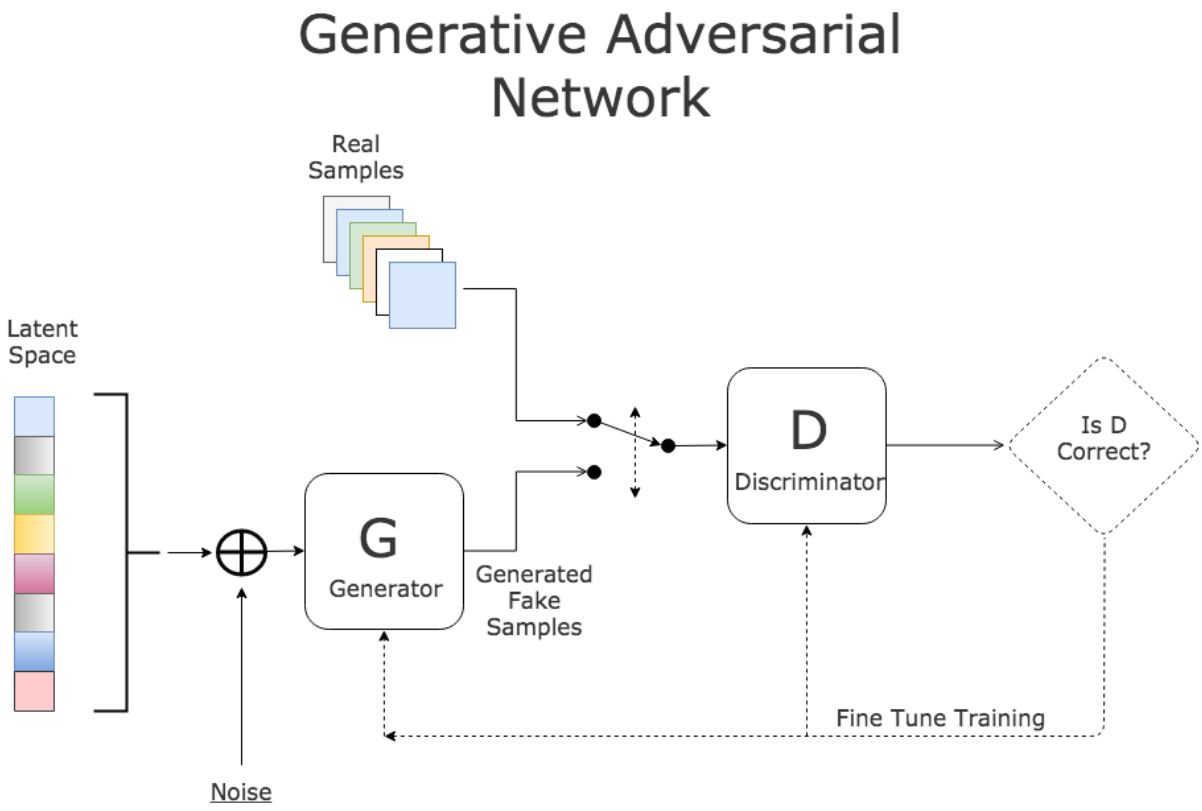


Figure 8: The process of GAN

Followed by a general description of the core principle of GAN, D and G play the following two-player minimax game with the value function V(G, D) (Goodfellow, 2014):

$$\min_G \max_D V(D,G)$$

$$V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z))]$$

- Similar to cross entropy in two-class classification problem in machine learning, the entire formula consists of two items, x stands for an actual image, z represents the noise image that will be input to G model, and G(z) stands for the result that noise image input to the G network.
- D(x) reveals the probability that the D model thinks x is true (because if x is true, so for D, it will be better if the value is closer to 1). D(G(z)) is the probability that the D model considers the G generated picture is true.
- The purpose of G, as mentioned above, is to create a picture that will cheat D and expects D(G(z)) to be as large as possible. If D(G(z)) is big, V(D, G) will become smaller. So, we see that the goal of the G is to minimize the formula(that's why min_G as the first symbol of formula).
- The purpose of D, on the other hand, is that if D gets stronger, the larger D(x) should be, and the smaller D(G(x)) should be. At this time, V(D,G) will increase. Therefore, the D should maximize the formula.

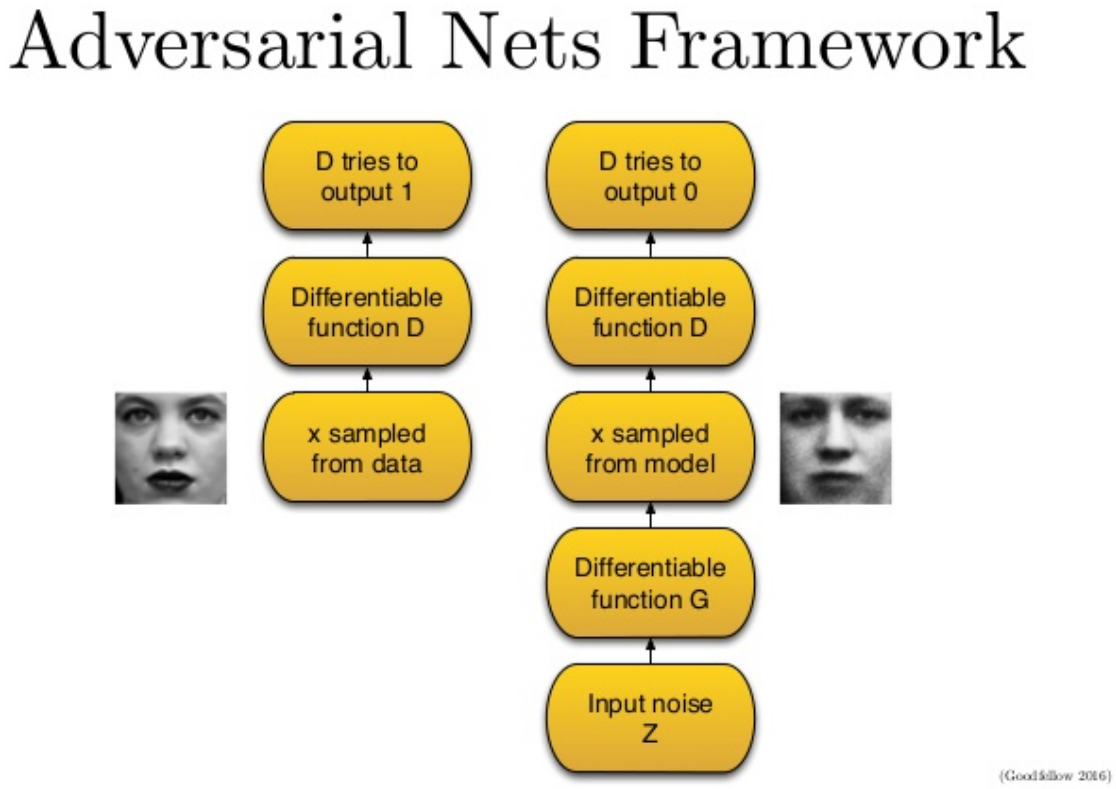


Figure 9: Adversarial Nets Framework(Goodfellow, 2016)

How can this maximum minimization objective function be optimized? The most intuitive approach is to interactively iterate D and G, fix G, and optimize D. After a period of time, fix D and optimize G until the process converges.

8.2 ResNet

The Deep Residual Network(ResNet) was a deep convolutional network proposed in 2015. Once it was born, it captured the image classification, detection, and positioning championships in ImageNet. ResNet takes a step further than convolutional neural network by doing the step-in figure 10.

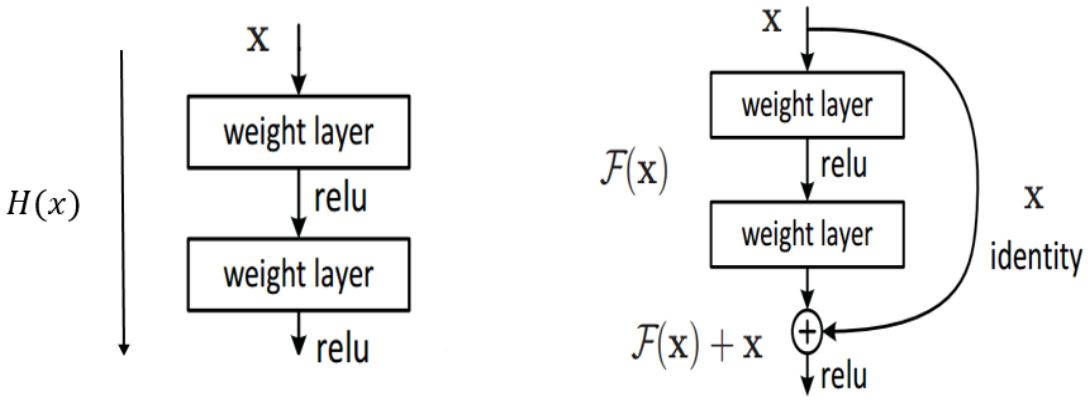


Figure 10: Normal CNN vs CNN with Residual Connection

For normal CNN, suppose we have an input $a^{[l]}$, then after the first weight layer, the output $z^{[l+1]} = W^{[l+1]}a^{[l]} + b^{[l+1]}$. After the activation function "ReLU", it turns out to be $a^{[l+1]} = g(z^{[l+1]})$. Similarly, in the second weight layer, $z^{[l+2]} = W^{[l+2]}a^{[l+1]} + b^{[l+2]}$, the final result is $a^{[l+2]} = g(z^{[l+2]})$. So, in other words, the information from $a^{[l]}$to $a^{[l+2]}$, it needs to go through all these step, which is called as "main path".

However, for residual connection, we are going to change it by adding a "short-cut" from $a^{[l]}$directly to $z^{[l+2]}$, before it applying to activation function. Now, $a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$. In the figure above, it is F(x)+x. The advantage of this shortcut is to optimize CNN by avoid gradient vanishing, which will be explained in next section.

The above two-layer component is called as "residual block". It will be used in our design in the latter explanation.

8.3 How GAN and ResNet contribute to image super-resolution

First, consider SRCNN, we want to create a better method based on SRCNN, one of the basic ideas is to increase the CNN layers because CNN can extract low/mid/high-level features. The more layers in the network, the richer features that can be extracted to different levels. If we get more features, image super-resolution performance will work better. However, there remains a question. For any neural network, if we simply increase the network depth(layers), it will lead to gradient vanishing or gradient explosion.

We know that for neural networks, we need to adjust the weight of the network through back propagation, like this:

$$Loss = F(X_L, W_L, b_L)$$

$$\frac{\partial Loss}{\partial X_L} = \frac{\partial F(X_L, W_L, b_L)}{\partial X_L}$$

But at this time, if the network is deep, it will lead to a negative issue:

$$Loss = F_N(X_{L_N}, W_{L_N}, b_{L_N})$$

$$L_N = F_{N-1}(X_{L_{N-1}}, W_{L_{N-1}}, b_{L_{N-1}})$$

$$\dots$$

$$L_2 = F_1(X_{L_1}, W_{L_1}, b_{L_1})$$

And when you do back-propagation to calculate its partial derivatives, it is:

$$\frac{\partial Loss}{\partial X_1} = \frac{\partial F_N(X_{L_N}, W_{L_N}, b_{L_N})}{\partial X_L} * \dots * \frac{F_2(X_{L_2}, W_{L_2}, b_{L_2})}{\partial X_1}$$

This value is what we want for the gradient. It is already very small and when we calculate it, we must multiply the step size and it is even smaller. Therefore, it can be seen that when we calculate the gradient in the back-propagation process, if N is large, then the gradient value will be smaller and smaller when it propagates to the previous layers, that is, the problem of the gradient vanishing.

However, in ResNet, what we have is $a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$. If we expand $z^{[l+2]}$, it is $a^{[l+2]} = g(W^{[l+2]}a^{[l+1]} + b^{[l+2]} + a^{[l]})$. If the gradient is zero, that is $w^{[l+2]} = 0$, $b^{[l+2]} = 0$, we still have $a^{[l+2]} = g(a^{[l]})$. But in previous CNN, everything vanished.
From what we demonstrated above, it is obvious that ResNet optimized and recovered the disadvantage of CNN by avoiding gradient vanishing or gradient explosion. Therefore, it quickly came to my mind that use ResNet must be a good way to optimize CNN.

Afterwards, we start to consider what GAN can do. GAN can generate an image by contesting the minimax game. Can GAN generate an image that is high-resolution? The answer is yes, but what we need to figure out is how to let HR image looks like LR one. Since we find some implicit connection between GAN and super-resolution, it is a promising job to explore super-resolution based on GAN.

9 Design

In this section, I will introduce my design of the project. Although we want to do 3D super-resolution, but what we do actually is on every stack of 2D images. Therefore, we should find a better way to get a perceptually good result. First, I am going to start by answer the question in the “Description of the Work” chapter, then I will show you the formula for the adversarial network. Once we know the formula of GAN, what we need to do is a detailed design of Generator Model and Discriminator Model. Afterwards, a discussion of perceptual loss function will be demonstrated.

9.1 How to get perceptually good result

Although we have used the faster and deeper convolutional neural network (CNN) to break through the speed and accuracy of single-image super-resolution, one central problem is still not perfectly solved: when super-resolution is applied to images that are multiplied by many times, how can we better restore the texture details of the image?

The super-resolution method based on the optimization idea is mainly driven by the objective function. Some recent related projects aimed at minimizing the average variance(MSE) reconstruction error. The result obtained has a large Peak Signal to Noise Ratio(PSNR), but often the image will miss high-frequency details and have poor visual effects. Therefore, unlike other previous super-resolution frameworks, SRGAN doesn’t use only pixel-wise mean square error in the loss function. To achieve the framework, it is proposed a perceptual loss function will be used, which consists of adversarial loss and content loss (Johnson, Alahi and Fei-Fei, 2018). The adversarial loss is generated by the discriminator, which makes the image we generate looks closer to the natural image. The content loss is generated by the visual similarity of the image, not the pixel space similarity. And the deep residual network(ResNet) can recover realistic textures from deeply down-sampled images.

9.2 Adversarial Network Formula

Before we actually figure out the network structure, we need to think about the formula of GAN, which is our ultimate goal. First, we need a lot of high-resolution images to train SRGAN network. I firstly down-samples the HR images to obtain LR images, then inputs the LR images and trains the generator model to generate a corresponding HR image. The process of training the generator is the same as training feed-forward CNN. Both of them optimize the network hyper-parameter θ_G, which including the weight w and bias b.

Assume, we have a perceptual loss function L^{SR}, the hyper-parameter θ_G should be trained by the following formula:

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

This formula shows that we need to find a parameter setting θ_G, that makes the loss function L^{SR} be minimum. The loss function L^{SR} takes two parameter, the first one is the image that generate by our G model, the other one is the original high-resolution image.

Recall the two-class cross entropy function of GAN:

$$\min_G \max_D V(D,G)$$

$$V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z))]$$

,

I define a min-max adversarial network function that we need to solve:

$$\min_{\theta_G} \max_{\theta_D} \mathbb{E}_{I^{HR}\sim p_{\text{train}}(I^{HR})}[\log D_{\theta_D}(I^{HR})] + \mathbb{E}_{I^{LR}\sim p_G(I^{LR})}[\log(1 - D_{\theta_D}(G_{\theta_G}(I^{LR})))] \qquad (2)$$

where D means the Discriminator model and it wants to maximize its ability to judge which image is high-resolution, whereas G means generator to try its best to lie discriminator.

9.3 Network Structure

The architecture of generator network consists of residual blocks and normal convolutional neural network. It is illustrated as follows in figure 11 (Arxiv.org, 2018):

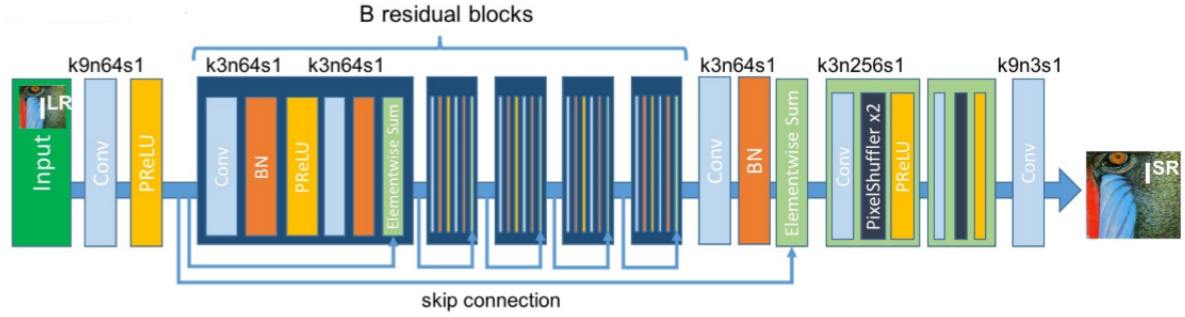


Figure 11: Generative Model Design

The generator network input a low-resolution image, usually in 112*112 size. It initially goes over a typical convolutional layer and a parametric rectifying linear unit(ReLU). Followed by a residual block, which has a batch normalization(BN) layer with a ReLU unit after the first one. According to Shi et. al., we increase the low-resolution image with two pre-trained sub-pixel

convolution layers. Then the output typically will be of size 224*224. You must be confused with the sequence of the letter above the network. K means the receptive field size(kernel size), n is the number of filters and s is the stride. This is the same as we did in normal CNN. On the other hand, the architecture of discriminator network is as follows:

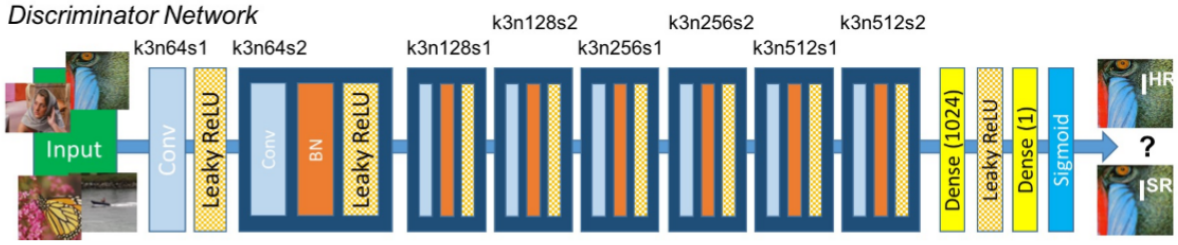


Figure 12: Discriminator Model Design

To avoid using the max-pooling layer in the network, we use a Leaky ReLU, which has all the advantages of ReLU but avoid "Dead ReLU Problem". Dead ReLU problem refers to the phenomenon that some neurons may never be activated, causing the corresponding parameters to never be updated. Normal ReLU = max(0, x), whereas Leaky ReLU =$max(\alpha x, x)$. Typically, alpha = 0.2. In the following figure, the left one is ReLU and the right one is Leaky ReLU.

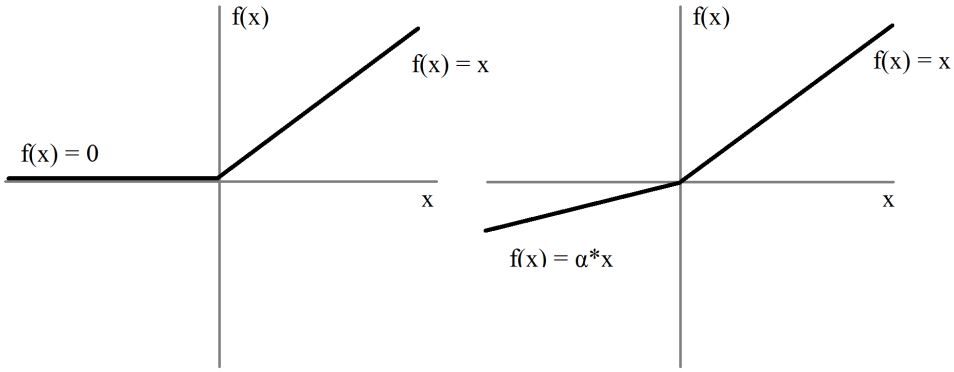


Figure 13: Traditional ReLU and leaky ReLU

What the discriminator network do is to solve the maximization problem in the above cross-entropy equation throughout the training process. There are eight convolutional layers, with the number of filters increasing from 64 to 128 to 256 to 512 as in VGG network. Finally, the resulting feature map has 512 in depth, this is followed by two dense layers and a sigmoid activation function, making it eventually converts into the probability of sample classification. However, in my implementation, I didn't add any batch normalization layer. The reason is that batch normalization is not suitable for super-resolution in my personal view. It helps to simplify and modify parameter setting because it is easier to convergence with normalized data. However, super-resolution is highly sensitive to the scale. The scale in image is a significant valid and effective information with respect to transformation of image. Actually, I also tried to add BN to CNN in my super-resolution work. However, it can be clearly seen from the training log that BN is totally unreasonable in super-resolution. The convergence rate of the network loss that uses BN is significantly slower than that without BN and the fluctuation of loss is very large. Therefore, adding BN increases the memory and computing load but has no prominent positive effect.

9.4 Perceptual Loss Function

As I mentioned above, the loss function of SRGAN should be slightly different from normal mean square error function. The perceptual loss is a loss function that compensates for missing details caused by MSE (Mean Square Error) and is modelled based on MSE. The perceptual loss function is defined as the sum of content loss and adversarial loss.

$$l^{SR} = \underbrace{\underbrace{l_{\mathrm{X}}^{SR}}_{\text{content loss}} + \underbrace{10^{-3} l_{Gen}^{SR}}_{\text{adversarial loss}}}_{\text{perceptual loss (for VGG based content losses)}} \tag{3}$$

9.4.1 Content Loss

As the formula (3) above illustrates, context loss is using perceptual similarity instead of pixel level similarity. Adversarial loss push SR image to the natural image manifold.

In the past, for example in SRCNN paper, we use MSE loss, we can still apply MSE loss in SRGAN, it is like:

$$l_{MSE}^{SR} = \frac{1}{r^2 WH} \sum_{x=1}^{rW} \sum_{y=1}^{rH} (I_{x,y}^{HR} - G_{\theta_G}(I^{LR})_{x,y})^2 \tag{4}$$

where W and H is the width and height of picture. r is the down-sampling factor, G_{θ_G} is the generator network as we mentioned above. It is a pixel-wise mean square error and it is easy for people to understand.

But we still propose a new Our content loss, instead, is like:

$$l_{VGG/i.j}^{SR} = \frac{1}{W_{i,j} H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y})^2 \tag{5}$$

where $\phi_{i,j}$ is feature map obtained by the j-th convolution after activation function and before the i-th max-pooling layer within the VGG19. $W_{i,j}\ and\ H_{i,j}$ are the dimension parameters of the feature maps.

9.4.2 Adversarial Loss

Except for content loss, other significant components in GAN is the generator. The effect of the generator on perpetual loss is reflected by the adversarial loss. This part of the loss function causes our network to "deceive" the discriminator so as to produce output that is closer to the natural image.

$$l_{Gen}^{SR} = \sum_{n=1}^{N} -\log D_{\theta_D}(G_{\theta_G}(I^{LR})) \tag{6}$$

Here, $D_{\theta_G}(G_{\theta_G}(I^{LR}))$ indicates the probability that the discriminator determines the image generator as a natural image, because if discriminator is good, it should be close to 1, therefore, $log\ D_{\theta_G}(G_{\theta_G}(I^{LR}))$ will close to 0. On the other hand, if discriminator is bad, it will close to 0, $log\ D_{\theta_G}(G_{\theta_G}(I^{LR}))$ will close to $-\infty$, and the loss will close to $+\infty$.

10 Implementation

In this section, I will introduce how I implement the whole algorithm. According to the time sequence, the implementation can be the pre-processing of 3D OCT and CT image, then using the extracted image to test our program. In terms of our program, I will first design a low-level utility python file including the most basic usage function. Then using the idea from OO programming, to develop higher level functions. Finally, I will reveal my challenges and problems encountered when I am implementing the program.

10.1 Pre-processing 3D OCT and CT image

In this section, I will show you the pre-processing of data in our project. Pre-processing refers to the processing after you received data from hospital, as we mentioned before, hospital use data in DICOM format. We use Matlab to change data into 2D image type like PNG format.

First, we need to get ".dcm" format. Typically, 3D OCT image will have ".img" file to describe image content and ".hdr" file as header file to describe the information. According to Matlab library, there are two functions to deal with it. Then we can have their content and assign them to ".dcm" file format. The content is a 3D image but what we do is to take out each stack of this 3D image. Each stack is a 2D image and we can simply do 2D image super-resolution on it. The code is shown in transfer.m as follows:

```
function [] = transfer(hdr_path,img_path)
    %read .hdr and .img file
    Info = analyze75info(hdr_path);
    Img = analyze75read(img_path);
    % Get the size of 3D image
    [m_height, m_width, m_thick] = size(Img);
    % Write dicom file, then we can use it in python(use pydicom)
    filename = '/Users/mikechen/Downloads/CT Image/CT_007_';
    % take each stack out
    % change name and save everything as .dcm file
    for i = 1:m_thick
        newname = [filename, int2str(i)];
        newname = [newname, '.dcm'];
        pp =Img(1:m_height,1:m_width,i);
        imshow(pp,[]);
        dicomwrite(pp,newname);
    end
end
```

Figure 14: transfer.m to extract image from hdr and img file

Since we already have all the ".dcm" format file, but what we want is an end-to-end implementation that input an image file and output the super-resolution one. Therefore, the next step in pre-processing is to change all the ".dcm" file to ".png" file.

To achieve this, we have a function called "dicomread", and we simply write the code for the program to automatically iterates every ".dcm" file in the folder and save them. The Matlab code for it is called "dcm2png.m" and can be found in tool folder.

If you run this Matlab code, you will find that the output of the image is the int16 format, but int16 cannot be written in ".png" format, that why we need to change it to uint16.

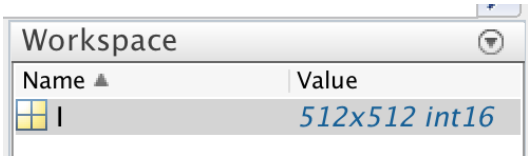


Figure 15: The format of Image I

Then we will have every ".dcm" to ".png":

IMG-0005-00001.dcm	2 Feb 2016 at 4:09 PM	527 KB	DICOM
IMG-0005-00001.png	13 Apr 2018 at 8:37 PM	85 KB	PNG image
IMG-0005-00002.dcm	2 Feb 2016 at 4:09 PM	527 KB	DICOM
IMG-0005-00002.png	13 Apr 2018 at 8:37 PM	83 KB	PNG image
IMG-0005-00003.dcm	2 Feb 2016 at 4:09 PM	527 KB	DICOM
IMG-0005-00003.png	13 Apr 2018 at 8:37 PM	82 KB	PNG image
IMG-0005-00004.dcm	2 Feb 2016 at 4:09 PM	527 KB	DICOM
IMG-0005-00004.png	13 Apr 2018 at 8:37 PM	79 KB	PNG image
IMG-0005-00005.dcm	2 Feb 2016 at 4:09 PM	527 KB	DICOM
IMG-0005-00005.png	13 Apr 2018 at 8:37 PM	78 KB	PNG image
IMG-0005-00006.dcm	2 Feb 2016 at 4:09 PM	527 KB	DICOM
IMG-0005-00006.png	13 Apr 2018 at 8:37 PM	79 KB	PNG image

Figure 16: Result after running dcm2png.m

10.2 Implementation of Algorithm

10.2.1 Platform Chosen

The project requires GPU machine acceleration to speed up the training time and training cost. Therefore, personal computer is not suitable. I bought a cloud server on AWS which use NVIDIA K80 GPU with 12 GiB memory and four CPUs. AWS stands for Amazon Web Service, which is the biggest on-demand cloud computing platforms to individuals. The AWS Deep Learning AMI provides machine learning practitioners and researchers with infrastructure and tools to facilitate deep learning at any scale in the cloud. You can quickly launch Amazon EC2 instances with pre-installed common deep learning frameworks to train complex custom AI models and experiment with new algorithms or learning new skills and techniques.

10.2.2 Language and Framework Chosen

The programming language I use is python. Python is the most prevalent programming language in the world. It is a high-level, interpreted dynamic programming language that focuses on code readability. It has fewer coding regulations but integrates extensive support libraries. The deep learning framework I will use is tensorflow. Tensorflow is an open source deep learning framework from Google. It can run on multiple CPUs and GPUs and available on any operating system and platform. Python has a support library for tensorflow, therefore, it is a perfect match for python and tensorflow. In terms of the library use in python, some of the essential libraries are numpy, opencv and math. I will not explain them in detail because I will focus more on the actual implementation ideology and steps. All networks are trained on NVIDIA K80 GPU, the GPU memory is 12GiB. However, to train the network, it still costs me 3 whole days to train, the process of training has 200000 iterations with the batch size of 16 and learning rate 10^{-4}.

10.2.3 Code Design and Writing

There are three aspects in terms of achieving the SRGAN. The first aspect is to do pre-processing/post-processing of your training data. Our training data is totally different from testing data, we use 8156 images from the RAISE dataset. Then we should have pre-processing such as load and save image. The second aspect is to implement your G and D model. The final aspect is to write some code to support command line arguments or scripts. I draw a diagram below to show all those aspect and detailed task to do in each aspect.

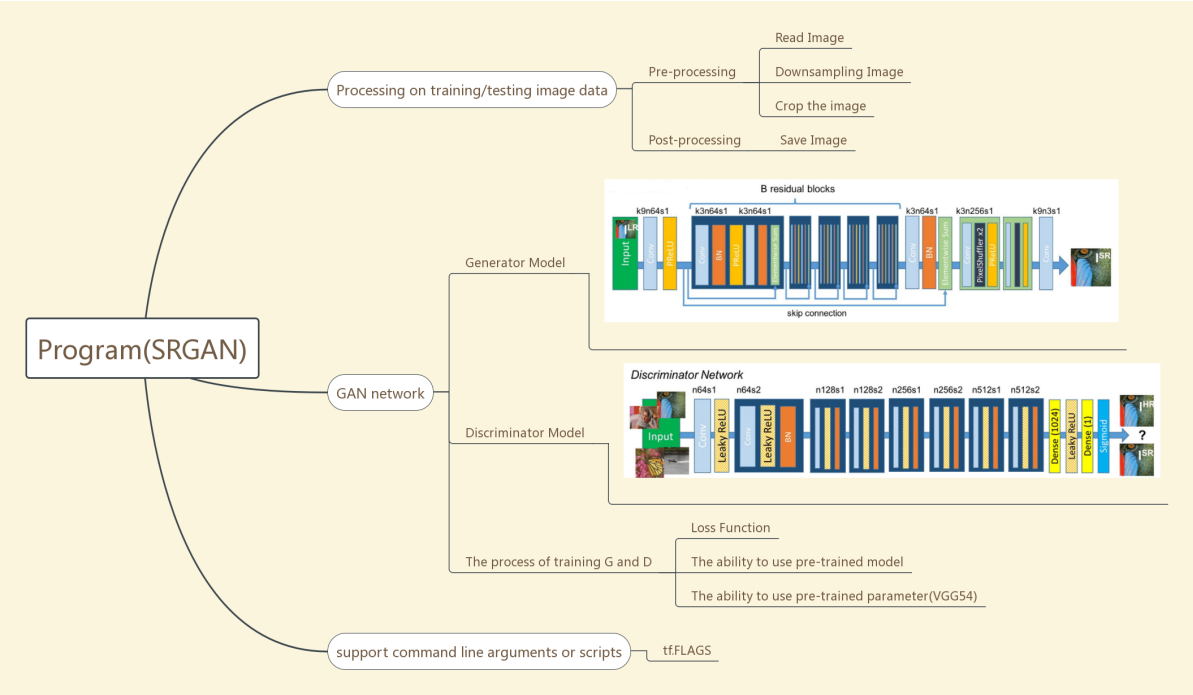


Figure 17: High-level code design

According to the figure 18 below, I write 3 main python file, called ops.py, model.py and main.py respectively. These 3 files are linked because I use the concept from object-oriented programming, to encapsulate some low-level function in ops.py and invoke them in model.py and also the highest level main.py.

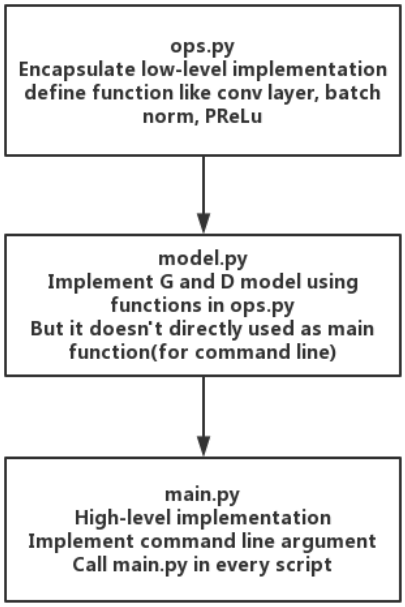


Figure 18: The hierarchy of python file

As I mentioned above, generator and discriminator actually consist of the basic element in deep learning, such as convolutional layer, batch normalization layer, PReLU activation function, dense layer and etc. Therefore, we should write a library file to encapsulate all the low-level implementation. Meanwhile, we can also write functions for pre/post-processing in ops.py, it is like:

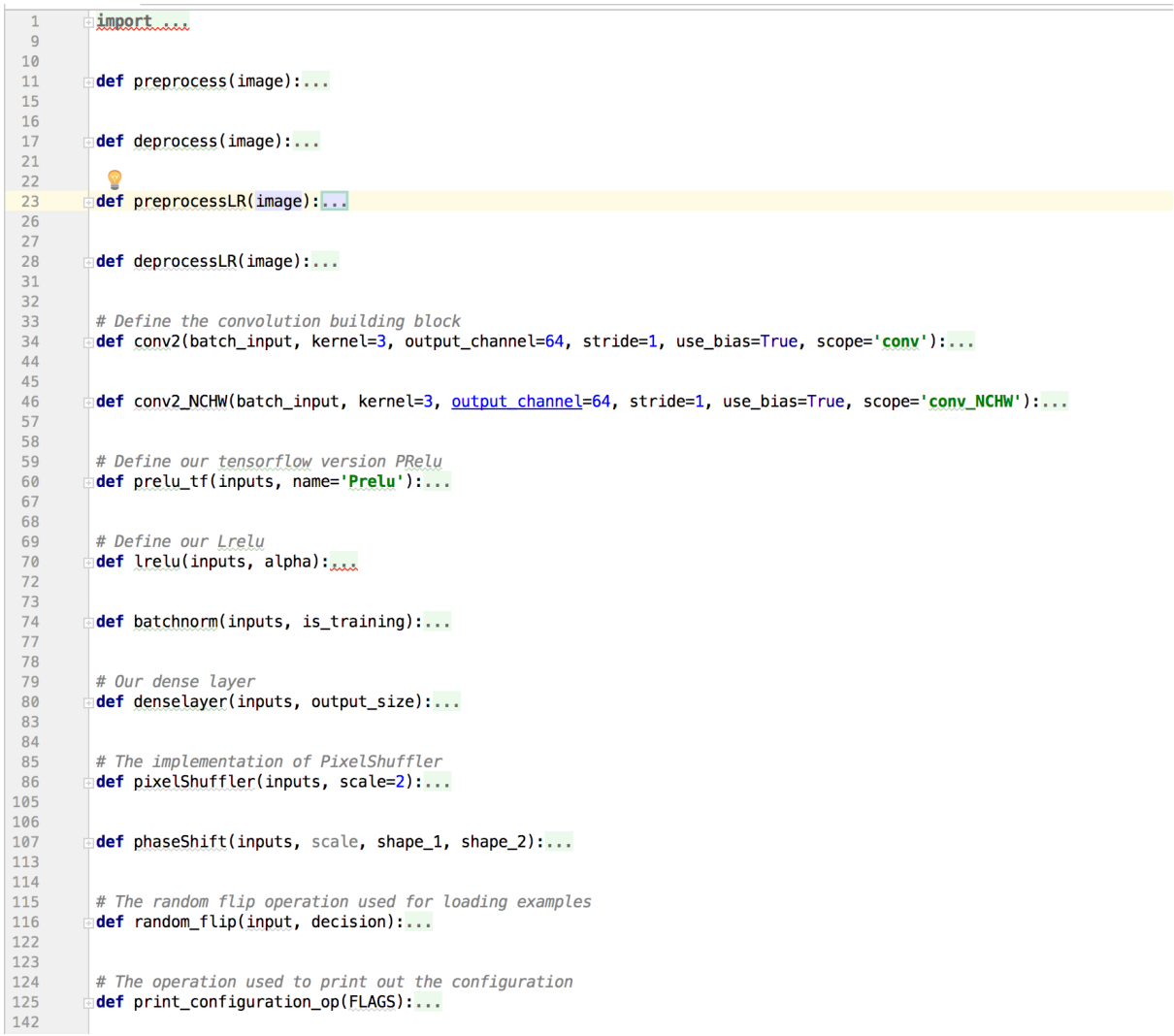


Figure 19: ops.py function definition

For example, if we want to write a convolution building block,

```
def conv2(batch_input, kernel=3, output_channel=64, stride=1, use_bias=True,
scope='conv'):
   # kernel: An integer specifying the width and height of the 2D convolution window
   with tf.variable_scope(scope):
       if use_bias:
           return slim.conv2d(batch_input, output_channel, [kernel, kernel], stride,
'SAME', data_format='NHWC', activation_fn=None, weights_initializer=tf.contrib.layers.xa-
vier_initializer())
       else:
           return slim.conv2d(batch_input, output_channel, [kernel, kernel], stride,
'SAME', data_format='NHWC', activation_fn=None, weights_initializer=tf.contrib.layers.xa-
vier_initializer(),biases_initializer=None)
```

The code actually use tensorflow internal function “conv2d”, specifying some appropriate parameter, such as the data format should be number of sample* height* width* channel. If the convolutional block use bias, we don’t need to initialize bias. Otherwise, we need to have bias initializer.

Since we have the low-level implementation, we can use them in our design of generator and discriminator. I use the same design as figure for generator, and ignore the “BN” layer in the discriminator.

```
def generator(gen_inputs, gen_output_channels, reuse=False, FLAGS=None):
def discriminator(dis_inputs, FLAGS=None):
```

These two functions implement the generator model and discriminator model. Then we have a function called “SRGAN(inputs, targets, FLAGS)” to achieve the whole network architecture. I first define a container of parameter:

```
Network = collections.namedtuple('Network', 'discrim_real_output, discrim_fake_output,
discrim_loss, discrim_grads_and_vars, adversarial_loss, content_loss, gen_grads_and_vars,
gen_output, train, global_step, learning_rate')
```

With so many parameter, we define each part of parameter, then we return the network structure as follows:

```
return Network(
    discrim_real_output = discrim_real_output,
    discrim_fake_output = discrim_fake_output,
    discrim_loss = exp_averager.average(discrim_loss),
    discrim_grads_and_vars = discrim_grads_and_vars,
    adversarial_loss = exp_averager.average(adversarial_loss),
    content_loss = exp_averager.average(content_loss),
    gen_grads_and_vars = gen_grads_and_vars,
    gen_output = gen_output,
    train = tf.group(update_loss, incr_global_step, gen_train),
    global_step = global_step,
    learning_rate = learning_rate
)
```

Figure 20: Detailed code implementation

Finally, we should have a high-level interface to execute the whole algorithm. I write a script to execute so we need to support command line argument. Tensorflow provides tf.app.flags to support arguments.

```
# The system parameter
Flags.DEFINE_string('output_dir', None, 'The output directory of the checkpoint')
Flags.DEFINE_string('summary_dir', None, 'The dirctory to output the summary')
Flags.DEFINE_string('mode', 'train', 'The mode of the model train, valid, test.')
Flags.DEFINE_string('checkpoint', None, 'If provided, the weight will be restored from the provided checkpoint')
Flags.DEFINE_boolean('pre_trained_model', False, 'If set True, the weight will be loaded but the global_step will still '
                                                 'be 0. If set False, you are going to continue the training. That is, '
                                                 'the global_step will be initiallized from the checkpoint, too')
Flags.DEFINE_string('pre_trained_model_type', 'SRResnet', 'The type of pretrained model (SRGAN or SRResnet)')
Flags.DEFINE_boolean('is_training', True, 'Training => True, Testing => False')
Flags.DEFINE_string('vgg_ckpt', './vgg19/vgg_19.ckpt', 'path to checkpoint file for the vgg19')
Flags.DEFINE_string('task', None, 'The task: SRGAN')
# The data preparing operation
Flags.DEFINE_integer('batch_size', 16, 'Batch size of the input batch')
Flags.DEFINE_string('input_dir_LR', None, 'The directory of the input resolution input data')
Flags.DEFINE_string('input_dir_HR', None, 'The directory of the high resolution input data')
Flags.DEFINE_boolean('flip', True, 'Whether random flip data augmentation is applied')
Flags.DEFINE_boolean('random_crop', True, 'Whether perform the random crop')
Flags.DEFINE_integer('crop_size', 24, 'The crop size of the training image')
Flags.DEFINE_integer('name_queue_capacity', 2048, 'The capacity of the filename queue (suggest large to ensure'
                                                  'enough random shuffle.')
Flags.DEFINE_integer('image_queue_capacity', 2048, 'The capacity of the image queue (suggest large to ensure'
                                                   'enough random shuffle')
Flags.DEFINE_integer('queue_thread', 10, 'The threads of the queue (More threads can speedup the training process.')
# Generator configuration
Flags.DEFINE_integer('num_resblock', 16, 'How many residual blocks are there in the generator')
# The content loss parameter
Flags.DEFINE_string('perceptual_mode', 'VGG54', 'The type of feature used in perceptual loss')
Flags.DEFINE_float('EPS', 1e-12, 'The eps added to prevent nan')
Flags.DEFINE_float('ratio', 0.001, 'The ratio between content loss and adversarial loss')
Flags.DEFINE_float('vgg_scaling', 0.0061, 'The scaling factor for the perceptual loss if using vgg perceptual loss')
# The training parameters
Flags.DEFINE_float('learning_rate', 0.0001, 'The learning rate for the network')
Flags.DEFINE_integer('decay_step', 500000, 'The steps needed to decay the learning rate')
Flags.DEFINE_float('decay_rate', 0.1, 'The decay rate of each decay step')
Flags.DEFINE_boolean('stair', False, 'Whether perform staircase decay. True => decay in discrete interval.')
Flags.DEFINE_float('beta', 0.9, 'The beta1 parameter for the Adam optimizer')
Flags.DEFINE_integer('max_epoch', None, 'The max epoch for the training')
Flags.DEFINE_integer('max_iter', 1000000, 'The max iteration of the training')
Flags.DEFINE_integer('display_freq', 20, 'The diplay frequency of the training process')
Flags.DEFINE_integer('summary_freq', 100, 'The frequency of writing summary')
Flags.DEFINE_integer('save_freq', 10000, 'The frequency of saving images')
```

Figure 21: Code to add parameters/arguments for python

I define a huge amount of parameters, including the input and output directory, the training or testing mode, the parameter in training and learning rate etc. After that, we can write code for each mode. The following code is the pseudocode for the main.py.

```
In [ ]: if FLAGS.mode == 'train':
            # load data and pre-process the data
            # create network, remember the function SRGAN(inputs,targets,FLAGS)
            # train it use MSE loss(previous method) or perceptual loss(the method in this dissertation)
            if loss == 'MSE':
                # define a loss function
            else
                # define VGG loss function
            # perform training
            for step in range(max_iteration):
                # save results
                results = sess.run(current_configuration);
                # save pre-trained model in checkpoint
                saver.save(sess, os.path.join(FLAGS.output_dir, 'model'))
        elif FLAGS.mode == "test"
            # declare the test data reader and read test image
            # declare the network
            # load pre-trained model
            weight_initializer.restore(sess, FLAGS.checkpoint)
            # start running
            for i in range(max_iteration):
                results = sess.run(current_configuration)
                save_image = save_images(results, FLAGS)
```

Figure 22: Pseudocode for main.py

A brief look at the code, you should aware that we first need to extract the mode from parameter/arguments. If the mode is train, we check which loss function you want to use, MSE loss or VGG loss(The difference of these loss is explained in 8.4.1). Then we can run the training process or testing process. If we train the model, we should be able to save the checkpoint

of training process or the ultimate model so that we can use the pre-trained model when we are testing. If we test the model, we should be able to save the generated image.

10.3 Train the model

To train the model, we need to add a lot of parameters when we run the main.py file. To simplify this work, I wrote some bash script. To train SRGAN, you should run the command "sh train_SRGAN" in the terminal. You can also check and modify the data and parameter for training. Training data and testing data are all from a data set called "RAW Image Dataset"(RAISE). Similar to ImageNet dataset, RAISE is a challenging real-world image dataset, which consists of 8156 high-resolution RAW images and low-resolution images. You can download RAISE from the internet for free. Before you trained the SRGAN, you can set all the parameter and other configurations in train_SRGAN.sh. In my personal experience, I use batch_size of 16 and max_iteration of 200000, and it costs me three whole days to finish the training on AWS GPU machine. When you train the network, you can be free to stop and resume. You can set how many iterations to log and record your network model and resume from any checkpoint.

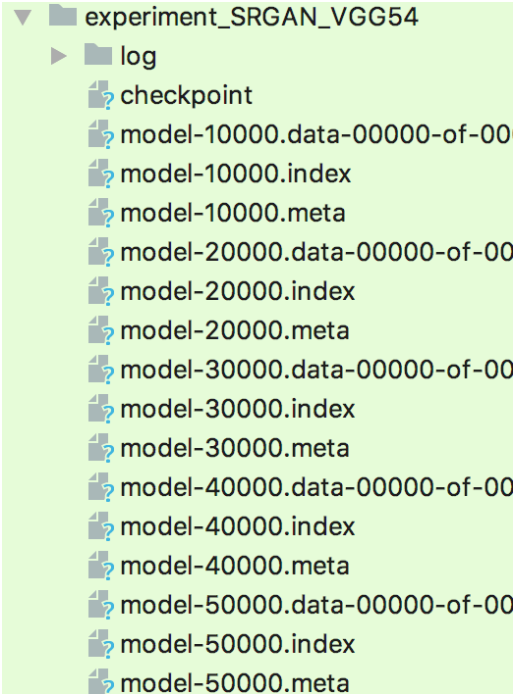


Figure 23: The screenshot of checkpoint saved in the training folder

It will take a long time to train. My environment for training is python 3.6, numpy 1.13 and Tensorflow 1.5, with GPU acceleration. The GPU is NVIDIA K80 and it still costs me 3 days to train the network. I have provided the pre-trained model after 200000 iterations training. During the training process, I wrote code for you to see the progress as the figure 24 shown below.

```
Recording summary!!
progress  iteration 119  step 120  image/sec 14.8  remaining 3608m
global_step 60300
PSNR 21.3072
discrim_loss 0.260808
adversarial_loss 5.19023
content_loss 0.0523625
learning_rate 0.0001
progress  iteration 119  step 140  image/sec 14.9  remaining 3573m
global_step 60320
PSNR 22.1775
discrim_loss 0.260538
adversarial_loss 5.13416
content_loss 0.0520144
learning_rate 0.0001
progress  iteration 119  step 160  image/sec 15.0  remaining 3539m
global_step 60340
PSNR 19.9926
discrim_loss 0.249982
adversarial_loss 5.14952
content_loss 0.0518312
learning_rate 0.0001
progress  iteration 119  step 180  image/sec 15.2  remaining 3511m
global_step 60360
PSNR 22.1816
discrim_loss 0.248438
adversarial_loss 5.13436
content_loss 0.0520366
learning_rate 0.0001
progress  iteration 119  step 200  image/sec 15.3  remaining 3484m
global_step 60380
PSNR 21.5743
discrim_loss 0.237774
adversarial_loss 5.09515
content_loss 0.051439
learning_rate 0.0001
```

Figure 24: The screenshot of training process, all the loss is decreasing

It is illustrated in the figure that the loss is decreasing all the time, which shows our implementation is correct.

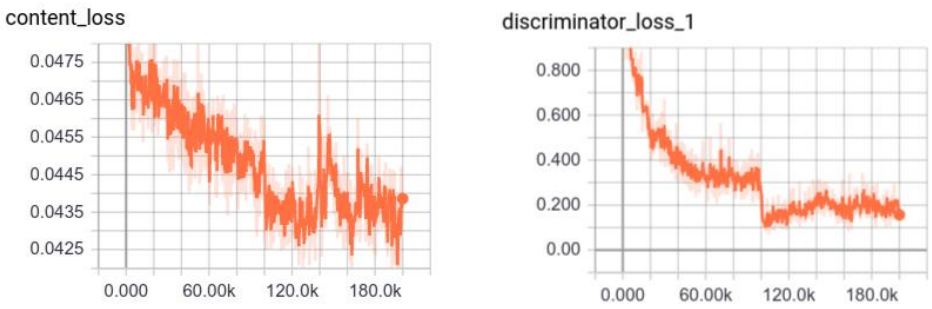


Figure 25: Use tensorboard to visualize loss

If you use TensorBoard API(tensorboard --logdir=path/to/log-directory) to draw the curve of content loss and discriminator loss, you will find both of them decrease till the end. And content loss increases a bit when it approaches 180000 iterations, which show it might overfitting afterwards. But it doesn't illustrate a strong overfitting according to the chart.

10.4 Challenges and problem encountered

While I am implementing my algorithm, I came up with a lot of challenges. I will explain some challenges I encountered.

The first challenge I encountered is I don't even know how to start my coding. Then I try to write some low-level utility functions and apply them into general GAN to check if it works.
When I started to create my function to read training data(images). I first want to use ImageNet data set. However, when I load them, I found the process is too slow. Then I found that the image is too big in size, what we should do is to crop the image. Then I found Tensorflow provides functions to crop the image. That's the why in model.py, there is code:
inputs = tf.image.crop_to_bounding_box(inputs, offset_h, offset_w, FLAGS.crop_size, FLAGS.crop_size)
In my personal perspective, it confused me long time, I should be aware of pre-processing image in the future.

The third challenge which makes me confused is also related to read image. Due to this problem, I stopped my coding for a week, even emailed my supervisor Bai Li for help. When I was testing my program, everything works well at first. When I try to test on a new image, it suddenly threw an exception that the image shape is not compatible. I tried to output the shape, it gave me (209,215,4). Then I am confused, because typically, the image we encounter is of three channels, that is RGB image. However, the shape of image shows that it is a four-channel image. After doing a lot of research, I found that it is actually a RGBA image, adding one parameter called alpha, which represents transparency. Then I wrote a function to transfer RGBA to RGB. Then I can read the image in but the colour is strange. The whole picture turns blue. Then after some time testing on different images, I found when you read image using python library called "scipy.misc", it will change RGB to RBG, by exchanging the position of channel. Finally, I found there is a better library called "python-opencv", which is a newly developed library for image reading and storing.

Another big challenge is about training. To start with my project, I work on my personal computer. I know it is slow with the only CPU to train the network but I have never thought how slow it is. The first time I try to train my network, it takes me one hour and the console print nothing. At that time, I think my program must stuck at some point, so I set a lot of break points to debug my program. Until then I found it stopped at a strange place where no possibility it could generate any bug. Then I try to use AWS with its high-speed GPU, I found the problem been solved.

Also, to simplified long-string argument, I wrote bash script as the interface. It is solution that enhance and optimize user experience.

11 Evaluation

This section will be divided into three parts. In the first part, I will show you a more human-sensitive way of evaluating image super-resolution. Then I will teach you how to test the algorithm. In the final part, I will give you a brief view of the result of the project using OCT and CT image.

11.1 Evaluation Criterion

In this dissertation, I will propose an innovative way of evaluating the quality of super-resolution. As we mentioned above, PSNR, previously, was considered as the evaluation criteria of super-resolution. PSNR refers to Peak signal-to-noise ratio, which can be calculate via MSE. $PSNR = 10 * log_{10}(\frac{MAX^2_I}{MSE}) = 20 * log_{10}(MAX_I) - 10 * log_{10}(MSE)$, where MAX_I means the maximum possible pixel value of the image. If it is an 8-bit image, the value is 255. Therefore, the greater PSNR value represents the less distortion. In this case, we have an original high-resolution image, and high PSNR corresponds to a closer similarity at the pixel-wise level. Then the higher PSNR value image shows that the result is outstanding. However, I mentioned in my question that "Imagine a pair of images, the second one copied the first, but changed a few pixels. For humans, replicas and originals are almost indistinguishable, but even such subtle changes can make the PSNR significantly lower". According to the figure 26 below, it is demonstrated that the first and second image has higher PSNR value than the third image. But it is obvious for human sense that the third image is clearer than the first one. Also, the figure 27 shows the curve of PSNR value when we are training the GAN network, it fluctuated in the range 21-24, but there is no clear trend of increasing in PSNR. Therefore, PSNR is acceptable but might not be the perfect evaluation criteria for image super-resolution.

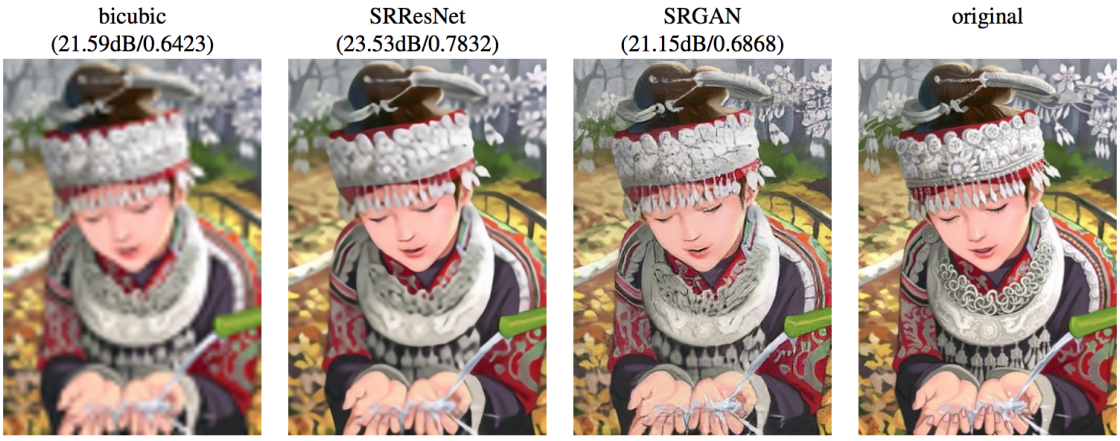


Figure 26: The comparison of PSNR value between bicubic, SRResNet and SRGAN

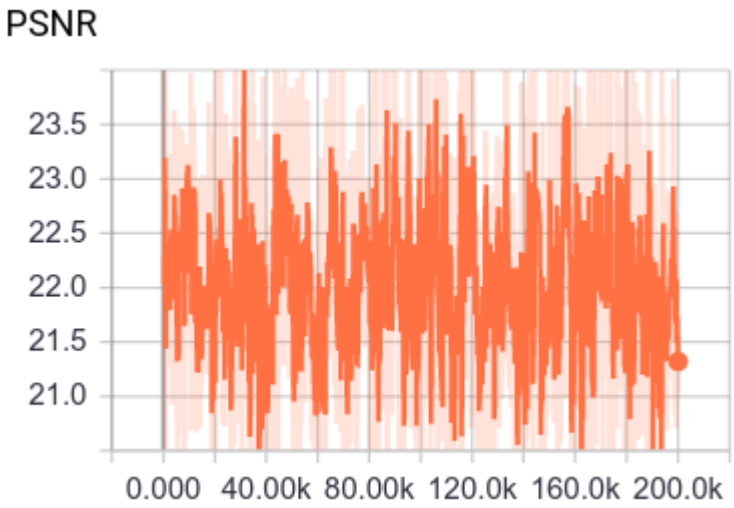


Figure 27: PSNR value during training SRGAN

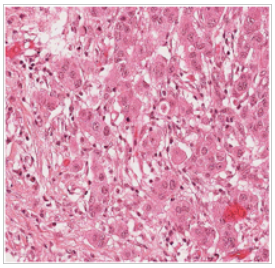
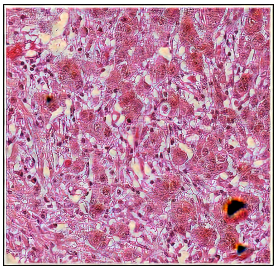
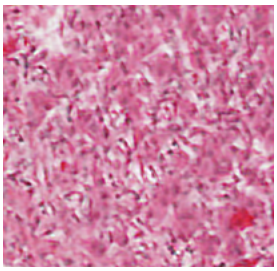
To substitute PSNR, the WQS(weighted quality score) is put forward as the evaluation of the image effect. We want to synthesize the pixel-wise similarity and human sensitive perception. It can be calculated as follows:

$$WQS = PSNR + \alpha HOS$$

The Human Opinion Score(HOS) can be obtained by investigation. 30 raters will judge which image is better. If there are several images generated by several methods, each people will rate each image on which method is better. And finally, we will sum up the number of favours of each method. Then we will follow the ranking table below, assume that there are 20 raters favours method 1 and the rest of 10 raters favours method 2. Then HOS(method1) = 8 and HOS(method2) = 4. Typically, the weighted value α = 0.3.

Number of favors	HOS
25-30	10
20-24	8
15-19	6
10-14	4
under 10	2

If we apply WQS as the evaluation criteria, the effect can be seen in the following table. It is obvious that the result from SRGAN is clearer than the result from SRCNN but the PSNR shows a higher value in SRCNN result. After the combination with HOS value, the final weighted score shows SRGAN is better.

Image	PNSR	Human Opinion Score(HOS)	Weighted Quality Score(WQS)
 Original Image			
 Result from SRGAN	23.88db	10	26.88
 Result from SRCNN	24.12db	2	24.72

I wrote a simple python code for calculating PSNR, which can be found in /lib folder. Then I use 20 OCT image as testing data set, applying SRCNN and SRGAN for them and the result is in the figure 28.

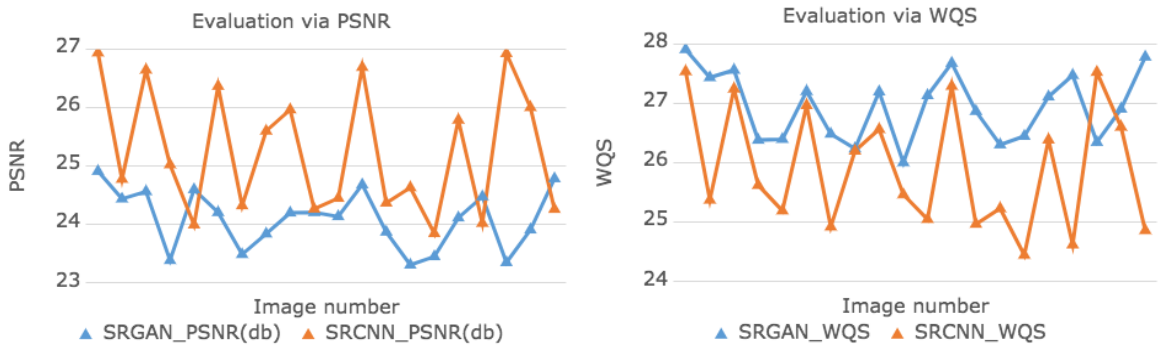


Figure 28: Line chart of PSNR and WQS value for SRGAN and SRCNN

According to the result, SRCNN is mostly higher in PSNR but lower in WQS. If we check the first line chart, blue points which stands for SRGAN's PSNR is mostly lower than SRCNN's PSNR. However, the second chart illustrates a total reverse opinion from the questionnaire outcome.

We should be aware that human's opinion is essential, especially when we applied it in OCT and CT images. Suppose, there is an OCT image, the one is doctor thinks better and the one is computer thinks. People would tend to believe doctor's view instead of cold-blood data numbers. What we perceive is more significant in the real-world situation, even more important than data.

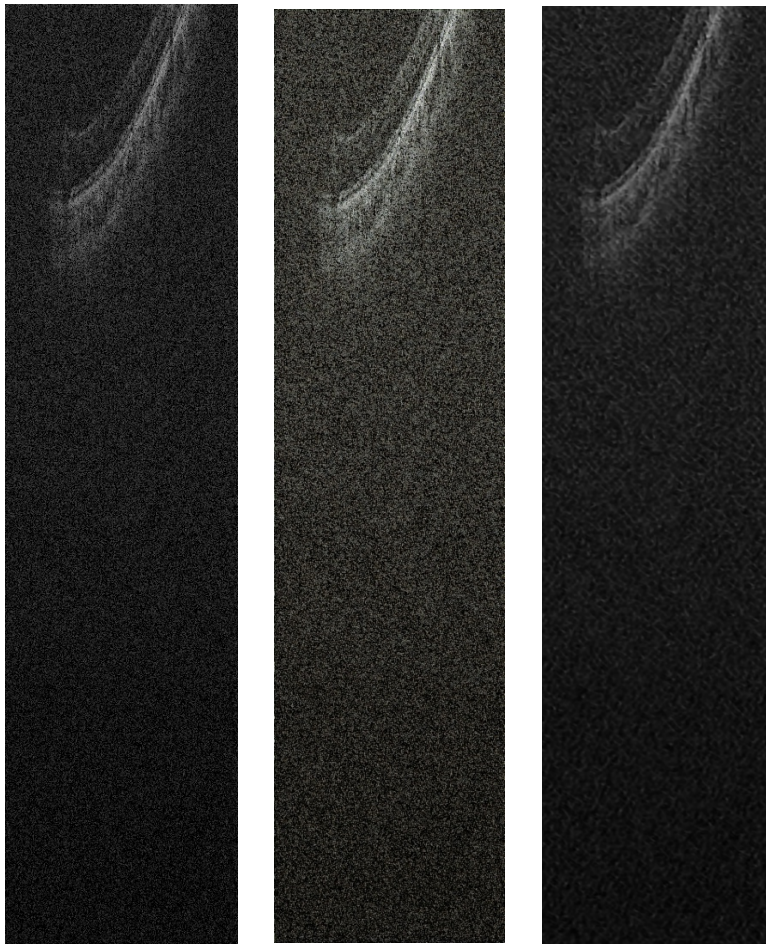
11.2 Testing process

To test the algorithm, I provide a test set of images, which are mostly from ImageNet dataset. After running "test_SRGAN.sh", you can find the result from result folder. Each image from test set will produce three images, which are the original image, low resolution one and the SRGAN-generated one.

11.3 A view of result

The OCT and CT image are 3D image, but I will show the outcome of one stack of 2D image.

11.3.1 OCT image



Original SRGAN SRCNN

Figure 29: From left to right is original image, SRGAN result and SRCNN result respectively

According to the result, you might feel this result is indistinguishable. However, SRGAN has two distinct benefits in the OCT picture. First of all, the SRGAN generated image is four times larger than the original image, and you will be able to see more details. Second, if you zoom in on each picture, you will see a slight change in the internal pixels.

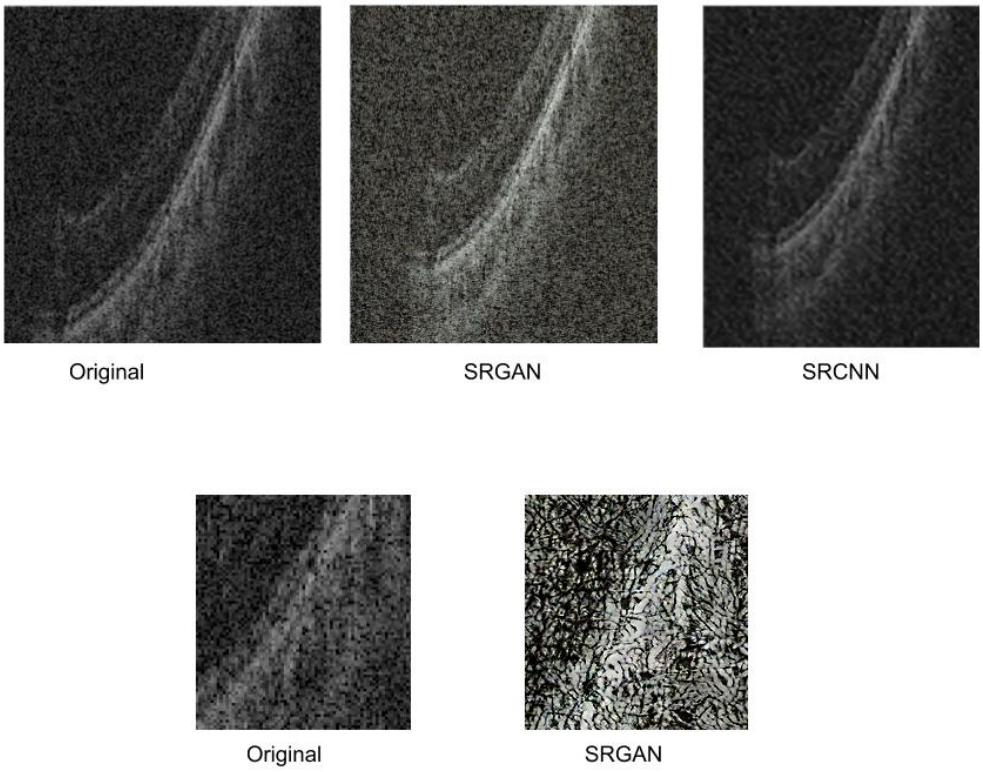
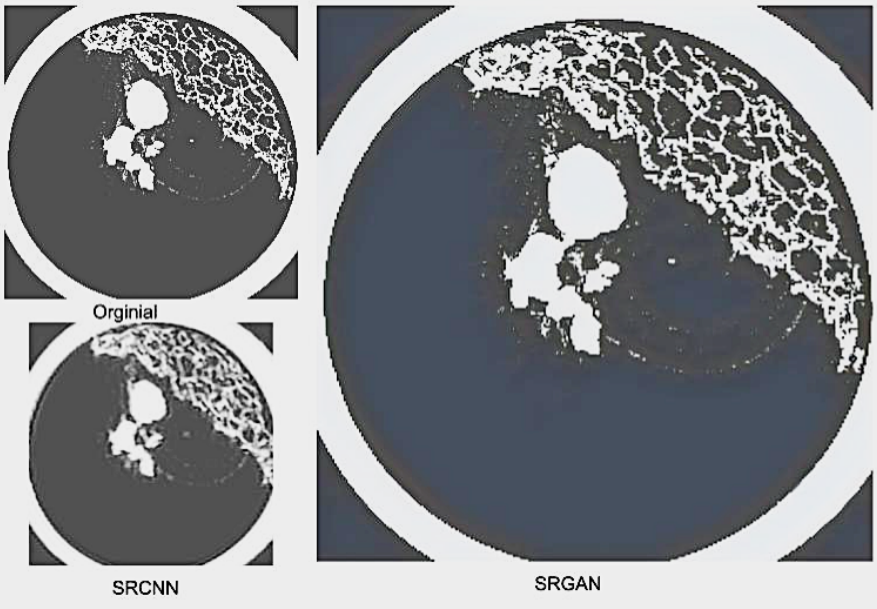


Figure 30: When you zoom in the image

From this white curve, it is obvious that the texture and particle characteristics more apparent.

11.3.2 CT Image



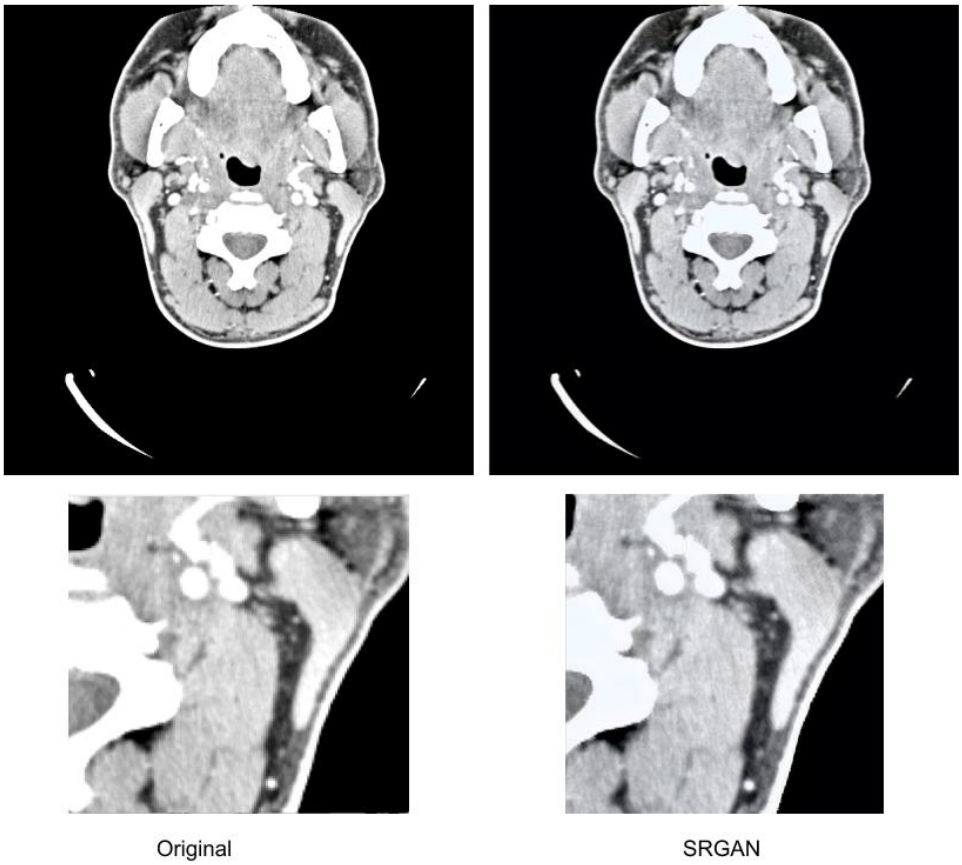


Figure 31: The result on CT images

The figure shows a more detailed view of SRGAN. First, SRGAN will upscale original image for 4 times. Second, if you zoom in, the texture and edge are more smooth in the SRGAN compared with the original image. For original image, if you zoom in a lot, you will soon see the image sawtooth and pixel blocks

12 Summary and Reflection

12.1 Project management covering and challenges encountered related to project

In my project proposal, I provided a Gantt chart for my project task management. Review this Gantt chart now, the overall work plan is finished in time and the project is well organized according to Gantt chart. However, there are some challenges and problem that I should notice in the future. Before I design the timetable and do the task management, I should be fully aware of the difficulty of the task. In this project, some tasks are too difficult to achieve in the required time period. I overestimated myself in coding, and it makes me suffered. Except for the implementation challenges mentioned above in 9.4, I also noticed some general issues related to project management. The majority of challenges are

1. I first implement SRCNN method, but actually the method is not outstanding in performance. How to figure out a more promising method is a big challenge for me.
2. When I get to know the basic knowledge of GAN, it is a problem that how it can be applied to super-resolution. What should be the neural network structure in the generator and discriminator

3. Coding in python is difficult as Tensorflow is an innovative area that I never learned and programmed before. I came up with a lot of coding challenges in python and figure them out by testing and google.

But I am also satisfied with my time management in this project. Even though I have various kinds of coursework to do in some time period, I am still able to fulfil the task and assign 15 hours on this project per week, even in examination period. In general, I think I got enough pressure during the working to push me forward, and I succeed.

12.2 Contributions and Reflections

In this section, I will analyze my contributions to this project and my reflections. The reflections will include my questions remained in this project and my personal reflections after experiencing the project.

My contributions are threefold. First, I analyze, investigate, classify and summarize the traditional super-resolution methods and state-of-the-art deep learning methods. Second, my work presents an innovative network structure for SRGAN and develop novel evaluation criteria for image super-resolution. Finally, using these methods, I make it on 3D images and compare the result of the current method with the prior methods.

In terms of the reflection, I still have some questions remained for this project. Due to the lack of time and ability, I can't figure them out. The first question is related to the network architecture. As I mentioned in Design part, I discard batch-normalization in discriminator model. However, in this year's Conference and Workshop on Neural Information Processing Systems(NIPS), there is a paper called "Self-Normalizing Neural Networks", it provided a new concept of self-normalizing neural network, which is used to substitute batch-normalization. But the paper is too long with 93 pages mathematical proof, I didn't understand it clearly. Will self-normalizing NN improve the performance of SRGAN still remains to be a problem.

The second question is about the loss function. When we calculate the content loss, we can choose to use MSE or VGG19 way of calculating them. But in $l^{SR}{}_{VGG(i,j)}$, it uses the Euclidean distance. However, what we calculate is the difference in feature map. Feature map is a high-dimensional space. What can be done to calculate the distance on high-dimension space remains to be a problem. It might cause curse of dimensionality.

The third question is that due to the characteristic of the generative adversarial network, the performance on several datasets is not satisfying. For example, you might see in the figure that the image is not good. There is some white colour space change to orange, which seems to be noise. This is the problem I can't figure out until the end of the project. This approach can be applied in most of the cases, but not all.

Also, due to the time and money cost on training, I have an idea that might improve and optimize the performance of the training model, but I didn't do that. My idea inherits from pre-trained model concept. When we use pre-trained VGG 19 networks with its weight, the performance will improve because the weight is better than those random initialization. So what if I trained my SRGAN with MSE first and use those weights as the initialization(first checkpoint) of the next training? Assume we have SRGAN trained with MSE, the result of

500000 iterations model, and we can train SRGAN with VGG loss again, but set the checkpoint as the MSE-result model. Will the performance be better? Theoretically it will but it will cost more time.

In terms of my personal reflection on this experience. It is my first time to do such a huge project by myself. It requires a lot of time to search the related literature on google, learn more state-of-the-art knowledge and concept, train the NN and adjust the parameter setting. While training the SRGAN, I need to spend 3 days to finish the whole training process. If the result is not what I expected, for example, overfitting, I need to modify the learning rate and train again. Furthermore, I also exercise my coding ability. Tensorflow and python are both useful tools for deep learning. Master these tools really help me in the future when I intend to do a project related to deep learning. Now I am able to build common neural network architecture. Other projects such as image style-transfer, image classification require the same requirements and I believe I can handle them.

12.3 Future Work

In the future, I will still work on this project because I am interested in solving problem related to image super-resolution. First, I want to solve the existing challenges and problems I put forward above, try to adapt the method to all kinds of CT and OCT images. Second, I intend to derive more extensions on 3D images. Also, in my code, I still reserve some code implementation for future use. For example, though I just do super-resolution based on GAN, I still set an argument called “task”. It means that if you want to add a new algorithm, assume SRXXX, you can simply add SRXXX’s code in model.py and alter your task in the script file. In this way, you can develop more super-resolution methods implementation on this code. Finally, I want to use current knowledge on super-resolution to figure out more fields such as real time video super-resolution.

13 Bibliography

[1] Arxiv.org. (2018). [online] Available at: https://arxiv.org/pdf/1609.04802.pdf [Accessed 21 Jan. 2018].

[2] Bhatt, Parth, Sachin Patel, and Rakesh Pandit. "Comparative Analysis of Interpolation and Texture Synthesis Method for Enhancing Image." International Journal Of Innovative Research In Science, Engineering And Technology 2 (2013)[Accessed 19 Oct. 2017].

[3] C. H. Pham, A. Ducournau, R. Fablet and F. Rousseau, "Brain MRI super-resolution using deep 3D convolutional networks," 2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017), Melbourne, VIC, 2017, pp. 197-200[Accessed 13 Jan. 2018].

[4] Demeyer, S. (2018). *Research Methods in Computer Science[Accessed 15 Apr. 2018]*.

[5] Dong, Chao, et al. "Image super-resolution using deep convolutional networks." IEEE transactions on pattern analysis and machine intelligence 38.2 (2016): 295-307[Accessed 21 Aug. 2017].

[6] "Deep Learning At The University Of Chicago." Deepdish.io. N.p., 2017. Web. 7 Dec. 2017.

[7] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. (2014). Generative Adversarial Networks. [online] Arxiv.org. Available at: https://arxiv.org/abs/1406.2661 [Accessed 20 Apr. 2018].

[8] Johnson, J., Alahi, A. and Fei-Fei, L. (2018). Perceptual Losses for Real-Time Style Transfer and Super-Resolution. [online] Arxiv.org. Available at: https://arxiv.org/abs/1603.08155 [Accessed 21 Feb. 2018].

[9] Kim, Kwang In, and Younghee Kwon. "Single-image super-resolution using sparse regression and natural image prior." IEEE transactions on pattern analysis and machine intelligence 32.6 (2010): 1127-1133[Accessed 10 Oct. 2017].

[10] Michal Irani, S. P., 1991. Improving resolution by image registration. *VGIP: Graphical models and image processing,* 1 5, 53(3), pp. 231-239[Accessed 5 Feb. 2018].

[11] Socscidiss.bham.ac.uk. (2018). *Research methodologies*. [online] Available at: http://www.socscidiss.bham.ac.uk/methodologies.html [Accessed 20 Mar. 2018].

[12] Tanno, Ryutaro, et al. "Bayesian Image Quality Transfer with CNNs: Exploring Uncertainty in DMRI Super-Resolution." Medical Image Computing and Computer Assisted Intervention – MICCAI 2017 Lecture Notes in Computer Science, 2017, pp. 611–619., doi:10.1007/978-3-319-66182-7_70[Accessed 19 Oct. 2017].

[13] W. Shi et al., "Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 1874-1883)[Accessed 19 Oct. 2017].

[14] Yang, Jianchao, et al. "Image super-resolution via sparse representation." IEEE transactions on image processing 19.11 (2010): 2861-2873)[Accessed 22 Aug. 2017].

[15] Zhaohui Zhang, Anran Liu, Qian Lei, "Image super-resolution reconstruction via RBM-based joint dictionary learning and sparse representation", Proc. SPIE 9815, MIPPR 2015: Remote Sensing Image Processing, Geographic Information Systems, and Other Applications, 981528 (14 December 2015); doi: 10.1117/12.2214097; http://dx.doi.org/10.1117/12.2214097)[Accessed 19 Oct. 2017].