“Deep Neural Networks for imaging in radio astronomy” by

Zeyu Xia

An Honours report submitted for the Degree of BEng in Electrical and Electronic Engineering

- Synopsis

Nowadays, Astronomers use radio telescopes arrays to explore the universe by sensing the radio waves which emitted by a wide range of objects such as sun or galaxy covering a vast swathe of the electromagnetic spectrum between around 30 MHz to 40 GHz.

The sensing modalities for telescopes arrays is to sample image signals in two-dimensional Fourier domain. However, the telescopes arrays can only measure some scattered points in the Fourier domain[1] owing to the interferometry sampling method in astronomy. Therefore, no matter how many antennas we have, we cannot fully cover the two-dimensional Fourier domain. In other words, the frequency information of the captured image is always incomplete.

The problem which tries to reconstruct ground truth pictures from incomplete information is called the reverse problem or the ill-posed problem. The reason we call it that is because recovering the original image from incomplete sampling information has an infinite number of solutions. However, only a few of them are natural images. Owing to this reason, not all of the inverse problems can be solved unless the images and sampling method meet certain conditions. The technique which can successfully reconstruct ground truth pictures from incomplete sampling information is called the compressed sensing[2].

In the past, reconstruction algorithms in compressed sensing are grounded in the field of optimization theory. Specifically, People construct a formula to minimize by combing a data fidelity term and a regularization term which contains the prior information of the image. The priori information guides us to find the solutions which are images among an infinite number of solutions by telling us the characteristics that the solution of the images satisfy. However, if the prior information is not accurate, the reconstruction effect will be unsatisfactory.

In recent years, deep learning techniques have become increasingly sophisticated. Mainly, Convolutional neural networks (CNN) show promising effects in image

recovery, such as denoising, deblurring [3]. What is more, the latest research results show that the prior information of image sets can be learned by training the denoising network[4]. These researches give us a new idea to solve the problem of the algorithm in optimization theory and get better reconstruction results in radio astronomy imaging.

- Acknowledgements

1. Sincere thanks for the guidance from:
   a) Prof. Yves Wiaux
   b) Postgraduate Research Tutor：Terris Matthieu
2. Thanks for the Pure Deep Neural Network result comes from:
   a) HaoXiang Sun

- Statement of Authorship

I, Zeyu Xia

State that this work submitted for assessment is my own and expressed in my own words. Any uses made within it of works of other authors in any form (e.g. Ideas, figures, text, tables) are properly acknowledged at their point of use. A list of the references employed is included.

Date.....16/04/2020.....

- Nomenclature

Symbol	Quantity
$\boldsymbol{\phi}$	Inverse sensing matrix
$\boldsymbol{\phi}^{\dagger}$	Sensing matrix
$\boldsymbol{\psi}$	Sparsity basis
$\boldsymbol{\psi}^{\dagger}$	Inverse Sparsity basis
$\boldsymbol{M}$	Selection mask
$\boldsymbol{m}$	Number of measurements
$\boldsymbol{x^o}$	Ground truth image
y	Sub-sampled data in the frequency
$\tilde{\boldsymbol{x}}$	dirty image$\tilde{\boldsymbol{x}} = \boldsymbol{F}(\mathbf{y})$
$\boldsymbol{F}$	discrete Flourier transform
$\boldsymbol{n_1}$	frequency domain noise with std$\boldsymbol{\sigma_1}$
$\boldsymbol{n_2}$	Time-domain noise with std$\boldsymbol{\sigma_2}$
$\boldsymbol{\rho}$	Penalty parameter
$\boldsymbol{\lambda}$	Regularization parameter
$\boldsymbol{\varepsilon}$	Noise boundary for $\boldsymbol{n_1}$

- Index

1 Introduction

1.1 Project Objection

The objective of this project is to implement a framework which plugs the deep learning technology into the algorithm of the classical optimization theory. We call this framework a hybrid algorithm. Besides, we will combine the theory with improving the performance of the hybrid algorithm gradually. Followed by that, we will also compare and analyze the hybrid algorithm with U-net[5], an end-to-end direct mapping neural network. At the end of the document, we will discuss the problems with the framework and point the way forward.

1.2 Description

At the beginning of the project, in chapter 2, we will go through a process from concrete problems to mathematical models. We will begin with a linear sampling model for astronomical imaging, discuss the conditions for sampling recovery with the help of the theory of compressed sensing and establish two objective functions based on the practical problem, namely, the constrained form and the unconstrained form. After that, we will discuss some algorithms for solving the objective function, with particular emphasis on convex optimization.

In Chapter 3, we will deduce the specific iteration form of the algorithm we will use in detail. We will start with the proximity operator, which is widely used in convex optimization problems. With the help of the proximity operator, we will deduce the concrete forms of the ADMM algorithm for two kinds of objective functions. Finally, we will update the ADMM algorithm to PnP algorithm with the help of the latest theoretical results.

Chapter 4 is about how to put theory into practice. In the first half of Chapter 4, we will discuss the implementation of the algorithm, including how to generate data, how to select parameters and the convergence of the algorithm. In the second half of

the fourth chapter, we will discuss the construction and training of a denoising neural network.

In Chapter 5, we will show the results. We will discuss some factors that affect the results to achieve a gradual improvement process. In the end, we will compare the hybrid algorithm with ADMM algorithm and pure neural network algorithm and discuss their advantages and disadvantages.

The last chapter, we will give the conclusion, and discuss the problem of the method, as well as the possible improvement scheme, in order to implement in the next stage.

1.3 Project Application

Our project itself stems from the practical problems of astronomical imaging. The theory behind the algorithm discussed in this project is based on compression sensing imaging technology. Therefore the methods discussed in this project can also be extended to the other field of compressed sensing imaging, such as MRI medical imaging.

2 Research background

2.1 Radio telescope sampling and linear sensing model

In this section, we will talk about the two characteristics of radio telescope sampling briefly according to the research in the astronomy imaging field. The detailed mathematic derivation of sampling characteristics is no our research emphasis. We will focus on constructing the operators and model to formulate the sampling process based on these two characteristics.

The first feature is that owing to the interferometry sampling method, the telescopes arrays are to sample image signals$x^o \in R^{N\times N}$ (see figure 1)in its two-dimensional Fourier domain(see figure 2) and get the sub-sampled Fourier coefficients $y^o \in C^{M\times N}\ with\ M < N$. For the operation which does sub-samples in the Fourier domain, we introduce a sensing matrix$\phi^\dagger = MF$ to represent. The symbol $F \in C^{N\times N}$ represents the discrete Flourier transform, and symbol $M \in C^{N\times N}$ is the selection mask which represents the measurement points sampled in the Fourier plane. In order to fit real sampling results, those points form an ellipse around the centre. (see figure 3).

The second feature is that the sampling process is linear. Specifically speaking, the x^o is sensed linearly by cross multiplying sensing matrix$\phi^\dagger$. Its projected value$\phi^\dagger x$, with the additive i.i.d Gaussian noise n form y together.

Therefore, we construct the linear sensing model to formulate this sampling process:

$$y = \phi^\dagger x^o + n\ \ with \phi^\dagger = MF$$

where y is the data in the frequency domain. we do the discrete inverse Fourier transform to y and get the time domain 'dirty image'$\tilde{x}$ (see figure 4) which is the untreated image that we captured. The reason why we call it 'dirty image' is that most of the texture embedded in high-frequency information disappeared and only a few of the bright stars embedded in low-frequency information exist. Besides, the

appearance of halo blurs the images. In the next subsection, we will talk about compressed sensing theory which can recover x^o from y.

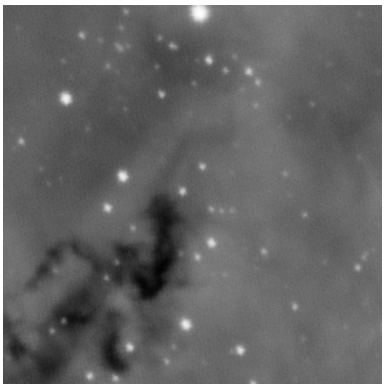


Figure 2 :Groundtruth image x^o

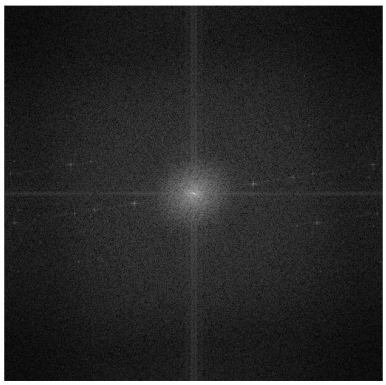


Figure 1 :**the two dimensional frequency domain**

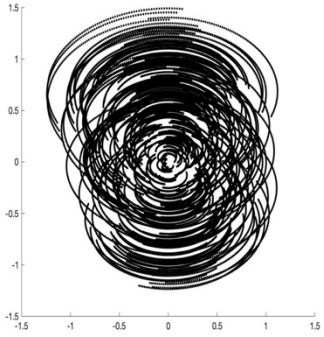


Figure 4: measurement points in frequency domain

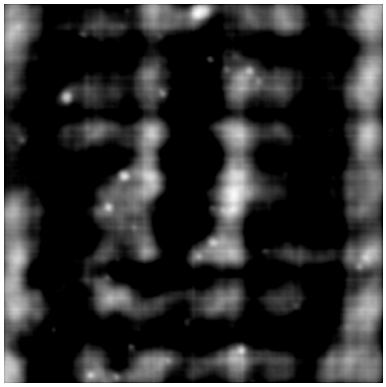
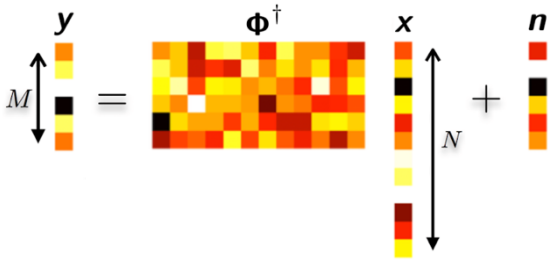


Figure 3:dirty image $\tilde{x}$

2.2 Compressed sensing

In this subsection, we will firstly talk about the difficulties that the reconstruction meets. Then we will introduce the compressed sensing theory. More specifically, we will talk about what is compressed sensing, and how can we use the theory of compressed sensing to solve the difficulties and guide our reconstruction.



[1]

Figure 5: Linear sensing model

For the ill-posed problem which tries to reconstruct ground truth pictures $x^o \in R^{n\times n}$ from sub-sampled results $y \in C^{m\times n}$, we need to solve the ill-posed equations system. For this kind of equations system, the number of equations m are less than the number of variables n which means that it has infinity number of solutions. However, only a few solutions which represent the natural image are the ones we want among the infinity number of solutions.

Fortunately, we can use the compressed sensing theory to solve difficulties that we meet. Compressed sensing is the technique which can successfully reconstruct ground truth pictures from incomplete sampling information [2]. However, not all of the ill-posed problem can be solved unless the images and sampling method meet certain conditions proposed by Compressed sensing. Imagine that, if an incomplete sample can be reconstructed well without satisfying the conditions, why do we still need Nyquist sampling?

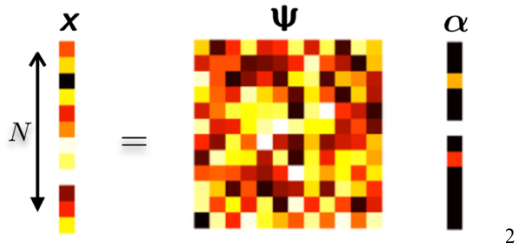
The first condition is that the image signal itself should be sparse in some linear transform domain, such as wavelet transform domain or Flourier transform domain. The linear transform operator takes the form of a matrix ψ. The sparsity can be expressed as:

$$x = \psi\alpha$$

[1] image stems from B31XO_2019-2020 Sampling and Computational imaging lectures

Where α is the sparse matrix which contains a small number of non-zero entries. If the number of non-zero entries is k among all of the components, we call this k-sparse matrix.

We can picture sparsity as follows:



[2]

Figure 6: Sparse model

Typically speaking, the wavelet transform base has a tremendous sparse expression result. Nearly most of the natural images are sparse on wavelet transform base. we can see the sparse degree for Figure_1:

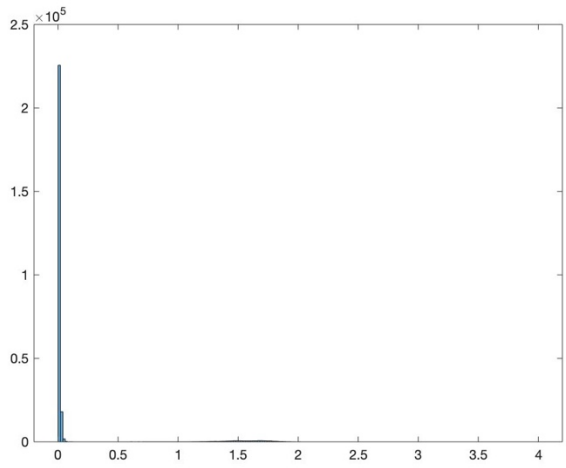


Figure 7: Sparse vector α

The second condition was given by Cand`es and Tao in 2005 [6]. They propose that the sensing matrix ϕ should satisfy Restricted Isometry Property (RIP):

$$(1-\delta_k)\|\alpha\|_2^2 \leq \|\phi\alpha\|_2^2 \leq (1+\delta_k)\|\alpha\|_2^2$$

Where α is the k-sparse vector andδ_k is the Restricted Isometry constant.

[2] Image stems from B31XO_2019-2020 Sampling and Computational imaging lectures

However, verifying sensing matrix satisfies RIP is complex. Baraniuk then proved that the equivalent condition of RIP is that the sensing matrix ϕ is sufficiently incoherent with the sparse representation basis ψ. The coherence is defined as:

$$\mu = \sqrt{n}\, max|< \phi_i, \psi_i >|$$

Then the question comes that for signal $x = \psi\alpha\,, x \in R^{n\times n}$ and with μ coherence and k sparse matrix α, at least how many measurements are needed to take in the Flourier domain of x so that we can recover the x in overwhelming probability ? Then, Tao derivate specific mathematical formulation for this question[7]:

$$Nnm\_mesurement \geq C \cdot \mu^2(\phi, \psi) \cdot (k * \log\, n)$$

According to this formulation, we can find that the higher coherence of the $\phi\ and\ \psi$, the more number of compulsory measurements are needed for reconstruction. In other words, For highly correlated $\phi\ and\ \psi$, it is the worst sensing method which would have a significant probability of missing a few non-zero entries unless we increase the number of the measurements.

It is worth mentioning that the location of these measurements points is reflected in the selection mask $M,$ which we discuss in the last subsection.

Once the conditions are met, we can construct the objective function and use the algorithm in compressed sensing to reconstruct ground truth pictures x^o from incomplete sampling information y. We will discuss these algorithms in the next subsection.

2.3 Objective function

In this section, we are going to construct our objective functions by the idea of shrinking the solution set step by step, and we are going to talk about two types of objective functions.

To solve an abstract problem, firstly, we need to construct the objective function. Following that, we can use the mathematic algorithm to solve it.

In the last section, we pay attention to the conditions that our image and sensing matrix should satisfy, and we have already known that most of the image signals are sparse in the transform domain, such as wavelet transform domain. The reason we emphasize this condition is that we will leverage sparsity to construct sparse regularization term in the objective function.

Regularization is a method which uses conditions to constraint the object. For sparse regularization term, it restricts the reconstruction result x should satisfy the sparse condition which we apriori know. As we said above, for an ill-posed problem, it has infinity number of solutions among which only a few solutions are natural image. Because noise and meaningless signals are not sparse in the transform domain. By using sparse regularization term, we can eliminate these types of signal from the solution so that we can find the natural image solution successfully.

Sparse regularization term $r(x)$ has the expression below. the value of ℓ_0 norm regularization is the number of non-zero parameters in sparsity basis $\psi^{\dagger}$.

$$r(x) = \left\|\psi^{\dagger} x_t\right\|_0$$

Now, the solutions set reduces to just the image solutions left in the set. this is good, but how to find the image which we want to recover?

By introducing a data fidelity term, we can solve this problem. The data fidelity term $h(x)$ has the expression:

$$h(x) = \left\|y - \phi^{\dagger} x\right\|_2 \; and \; h(x) \leq \varepsilon$$

Where $\varepsilon > 0$ is a known noise boundary on the corresponding noise level.

Before we talk about this expression, we should know that the L2 norm measures the 'distance' or difference between two vectors. Actually, the Euclidean distance is a kind of L2 norm.

From this perspective, we can see that $\left\|y - \phi^{\dagger} x\right\|_2$ measures the 'distance' between the sampling information y and the solution we search in the solutions set.

According to the linear sensing model $y = \phi^{\dagger} x^{o} + n$ that we discuss in the 2.1

section, because of the noise exists, $\|y - \phi^{\dagger}x\|_2$ will inevitably have the 'distance'. However, this 'distance' will be smaller than the noise boundary ε that we estimate corresponding to the different noise level. If this 'distance' exceeds the boundary ε, then it is not the solution we are trying to recover from the sample value.

We will use a diagram to describe the process of building the objective function：

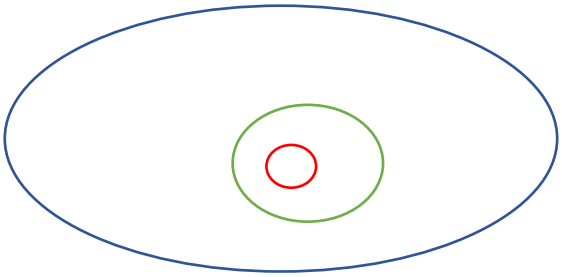


Figure 8: Approximation of solution set

The blue line represents infinity solutions set of the ill-posed problem. We then use the sparse regularization term to decrease the infinity solutions set to the image solutions set (green line). After that, we use data fidelity term to bound the solution x^* we want (red line).

Now our objective function has the expression below. We would minimize the signal sparsity subject to the data fidelity.

$$x^* = arg\,min\{r(x)\} \quad subject\ to \quad h(x) \leq \varepsilon$$

$$x^* = arg\,min\left\{\|\psi^{\dagger}x_t\|_0\right\} \quad subject\ to \quad \|y - \phi^{\dagger}x\|_2 \leq \varepsilon$$

However, the convex optimization algorithm does not apply to our objective function, because L0 norm is a non-convex term. An exhaustive search would be needed to find the solution. In other words, this an NP-hard problem.

Fortunately, Donoho et al. have proposed that L1 norm is the tightest convex relaxation of L0 norm[8] which means that We can replace L0 norm by L1 norm to make equation convex and avoid an np-hard problem.(the reason we will discuss in next convex optimization algorithm section) Now, our objective function has the following expression. Owing to the existence of constraint term, we call it constrained analysis objective function:

$$x^* = arg\,min\left\{\left\|\psi^\dagger x_t\right\|_1\right\} \quad subject\ to\ \ \left\|y - \phi^\dagger x\right\|_2 \le \varepsilon$$

However, it is still not convenient to apply the convex optimization algorithm to this form, because we have not expressed the constrained relation mathematically. Therefore, we can introduce an indicator function to replace the constraint term. the indicator function is defined as:

$$l_{B(c,\varepsilon)}(u) = \begin{cases} 0 & \|u - c\|_2 \le \varepsilon \\ +\infty & otherwise \end{cases}$$

Then the constraint term can be expressed as:

$$l_{B(0,\varepsilon)}(y - \phi^\dagger x) = \begin{cases} 0 & \left\|y - \phi^\dagger x\right\|_2 \le \varepsilon \\ +\infty & otherwise \end{cases}$$

For the indicator function $l_{B(0,\varepsilon)}(y - \phi^\dagger x)$, if $y - \phi^\dagger x$ does not satisfy the constrain $\left\|y - \phi^\dagger x\right\|_2 \le \varepsilon$, then the value of the indicator function will become infinity which means that this solution x cannot be the minimum point.

By introducing the indicator function, our constrained analysis objective function changes to the following form, which we use most often:

$$x^* = arg\,min\left\{\left\|\psi^\dagger x\right\|_1 + l_{B(0,\varepsilon)}(y - \phi^\dagger x)\right\} \qquad x \in C^N$$

There is another kind of objective function called unconstrained analysis objective function, which is the variant of the constrained analysis objective function. It is applied when we do not know the noise level and noise boundary ε. Its specific form is as follows:

$$x = arg\,mi\,n\left\{\lambda\left\|\psi^\dagger x\right\|_1 + \frac{1}{2}\left\|y - \phi^\dagger x\right\|_2^2\right\} \quad x \in C^N$$

we can see that it also contains two parts. Regularization terms is $r(x) = \left\|\psi^\dagger x\right\|_1$ which is same as the constrained analysis objective function. However, for the data fidelity term, owing to the unknow noise boundary ε, we use the L2 norm squared term $h(x) = \frac{1}{2}\left\|y - \phi^\dagger x\right\|_2^2$ as the data fidelity term. In order to balance the weight between the fidelity term and regularization term, we also need to introduce the

regularization parameter λ. Usually, this free parameter is sensitive and needs to be fine-tuned.

Up to now, we have constructed unconstrained and constrained objective function. In the next sub-section, we will briefly discuss which classical algorithms can be used to solve the target equation, and we will discuss how these classical algorithms can be combined with the current hot topic of deep neural network.

2.4 Reconstruction algorithms and related work

At present, the reconstruction algorithm of compressed sensing is mainly divided into two categories. One is the greedy algorithm, and the other is the convex optimization algorithm.

Greedy algorithms are referred as some kinds algorithms that have the same kind of idea which is to decompose sampled signal y into a sparse matrix α (atoms) based on the measurement matrix (dictionary) $A = \phi^{\dagger}\psi$. we retroactively extrapolate the sampled signal y from the sparse matrix α (atoms) and changing the value of the α continuously to make the residual $y - \phi^{\dagger}\psi\,\alpha$ smaller[9]. Then we can use sparse matrix α to reconstruct ground-truth signal $x = \psi\alpha$. Since the greedy algorithm is not the focus of this project, we will not discuss it detailly. The standard greedy algorithms include the Matching Pursuit algorithm (MP) and the orthogonal Matching Pursuit algorithm (OMP). Compared with the convex optimization algorithm, The advantages of the greedy algorithm are low complexity and less computation. However, it also has a lower solution precision.[10]

After a brief discussion of the greedy algorithm, the convex optimization algorithm discussed next is the focus of this project. Recall that in the last section, we discussed L0 norm is the non-convex term. In order to avoid the NP-hard problem, we replace the L0 norm by the convex L1 norm. Why can we avoid NP-hard problems by making the equation convex？ To answer this question, we will first introduce what

‘convex’ is, then what is an optimization problem. Finally, we will discuss what advantages convex functions have that non-convex functions do not have for an optimization problem. When we are through with these discussions, the above problems are easily solved.

let us start with the definition of convex functions. the convex function has the Mathematical expression[11]:

$f\ is\ called\ convex\ funcion\ if$

$\forall x_1, x_2 \in X, \forall t \in [0,1]: \quad f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$

The Mathematical explanation for this expression is:

If a real-valued function f is a convex function, then the line segment between any two points on the graph of the function lies above or on the graph.

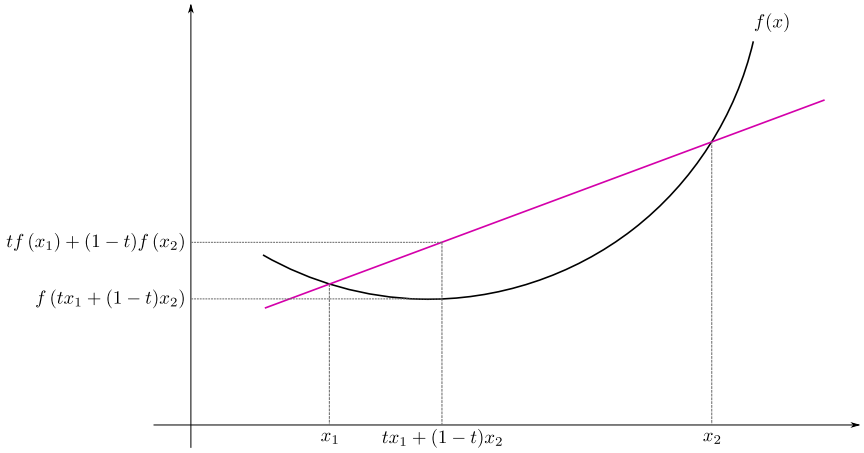


Figure 9: illustration of definition for a convex function

3

From the following picture, we can clearly judge that the L1 norm is a convex function and the L0 norm is a non-convex function:

[3] By Eli Osherovich - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=10764763

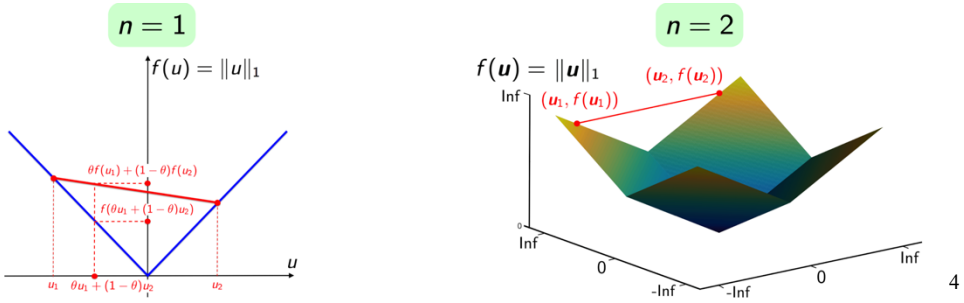


Figure 10: $\ell_1$norm is convex

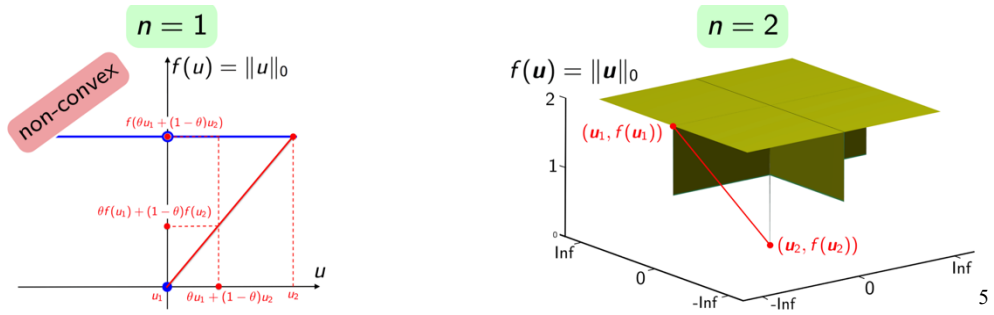


Figure 11:ℓ_2 norm is non-convex

Now that we understand the definition of what is convex, let us talk about what is an optimization problem. For optimization problem is a problem which consists in minimizing a function $f(u) = r(u) + h(u)$ over a set C:

$$minf(u)\ ,\ u \epsilon C$$

$$\min\{r(u) + h(u)\},\ u \epsilon C$$

In other words, we want to find a minimum $u^* \in argmin\, f(u)\ ,\ u^* \epsilon C$

The convex optimization problem is a particular case when C is a convex set and f is a convex function. It has essential properties that can be applied to the find the minimum:

For differential and convex object function, the necessary and sufficient condition for $u^*$to be the minimum is that the derivative at $u^*$is zero $\nabla f(u^*) = 0$.

---

[4] *image stems from B31XO_2019-2020 Sampling and Computational imaging lectures*

[5] *image stems from B31XO_2019-2020 Sampling and Computational imaging lectures*

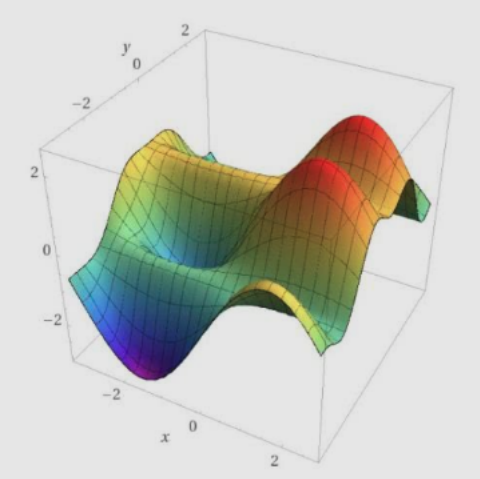
For non-differential and convex object function, the necessary and sufficient condition for u^* to be the minimum is that the sub-gradient at u^* contains zero-point $0 \in \partial f(u^*)$.

The reason why these two properties are so important is that they relate points with a gradient or subgradient of zero to the minimum. Specifically speaking, for differential and convex object function, it is obvious that we can calculate the zero-gradient point to find the minimum directly. Besides, if there is no closed-form solution or it is too expensive to compute, then we can get iteratively approach to the point where the gradient is zero by making a small step in the opposite direction of the gradient. This algorithm is famously known as gradient descent[12] which has the following expression. We will not discuss this algorithm detailly.

$$x_{n+1} = x_n - t * \nabla f(x_n)$$

On the other hand, for non-differential and convex object function, we can use the property to construct conditions for obtaining optimal solutions, such as the proximal operator of ℓ_1 norm which we will discuss in section 3.

However, such significant properties for non-convex optimization problem are not available. $\nabla f(u^*) = 0$ or $0 \in \partial f(u^*)$ is a necessary but not sufficient condition for u^* is a minimum since they have a great number of or even infinity number of zero-gradient points which may be local minimum or local maximum even a saddle point. We can see this clearly from the picture below.



[6]

Figure 12: Non-convex function

Therefore, for the non-convex optimization problem, It is not enough to exhaustedly find the zero-gradient points， we also need to compare the values at these zero-gradient points. As the number of zero-gradient points goes to infinity (L0 norm is also an example that all of the points except the origin of coordinates have the zero gradients), this non-convex optimization problem becomes an NP-hard problem.

Now, we know the benefits of convex optimization problems. And then we are going to focus on what are the algorithms that we can use to solve our objective functions in convex optimization fields.

Recall that we have two objective functions, for constrained analysis objective function:

$$x^* = arg\,min\left\{\left\|\psi^\dagger x\right\|_1 + l_{B(0,\varepsilon)}(y - \phi^\dagger x)\right\} \qquad x \in C^N$$

For unconstrained analysis objective function:

$$x = arg\,mi\,n\left\{\lambda\left\|\psi^\dagger x\right\|_1 + \frac{1}{2}\left\|y - \phi^\dagger x\right\|_2^2\right\} \quad x \in C^N$$

[6] *image stems Wikipedia of convex optimization topic*

We can find that both of the objective functions are not differentiable owing to the L1-norm, which is not continuously differentiable at the origin of coordinates. For non-differential and convex object function, the Lagrange multiplier method can be used to solve this kind of problem. By combing the Alternating Direction Method Of Multipliers (ADMM)[13], we can block the algorithm into two sub minimization problems which are effortless to solve. We will detailly discuss these implements in section 3.2 and 3.3.

In recent years, some hybrid algorithms which are based on the ADMM have been proposed. The two most famous hybrid algorithms are plug and play prior (PnP) [4]and Regularization by Denoising (RED)[14]. Plug and play prior (PnP) advocate using denoising network replacing proximal operator of regularization term and Regularization by Denoising (RED) propose a new regularization term which combines the denoising network. Although their algorithm forms are different, their core idea is the same, which is to use the denoising network to express priori instead of sparse regularization term. We will discuss the PnP hybrid algorithm in 3.4 and 3.5.

3 Methodology

In this chapter, because our objective function is non-differentiable, we will firstly discuss the proximal operator and Proximal Gradient Descent which is widely used in convex optimization of the non-differential objective function. After that, we will derive the iterative form of the Alternating Direction Method of Multipliers (ADMM) algorithm based on Augmented Lagrange Multipliers (ALM) and derive the corresponding PnP hybrid algorithm according to the latest research results. Finally, we will discuss the methodology of the pure neural network mapping method in imaging inverse problems.

3.1 Proximal operator and Proximal Gradient Descent.

In this sub section, we will introduce proximal operator which is a widely used operator in non-differential convex optimization problem. Then we will introduce the proximal gradient descent method which is a lightweight algorithm aimed to solve non-differential convex optimization. This proximal gradient descent method will be used to solve the sub-question of ADMM which we talk about in next sub section.

Remember that we discussed two essential properties of convex optimization in the previous chapter. One of the properties is for differential and convex object function $\boldsymbol{f(u) = r(u) + h(u)}$, the necessary and sufficient condition for $\boldsymbol{u^*}$to be the minimum is that the derivative at $\boldsymbol{u^*}$ is zero $\boldsymbol{\nabla f(u^*) = 0}$. A famous algorithm Gradient Descent, which iteratively approach to the point where the gradient is zero by making a small step in the opposite direction of the gradient, is based on this property. The condition for object function $\boldsymbol{f(u)}$ to be differentiable is that both of the components $\boldsymbol{r(u) + h(u)}$ must be differentiable. If one of the $\boldsymbol{r(u)\ and\ h(u)}$ is not differentiable which leads the objective function $\boldsymbol{f(u)}$ is not differentiable, then how to solve the convex optimization problem of such a non-differentiable

objective function? To answer this question, we first introduce the proximal operator.

When dealing with non-differentiable objective function in convex optimization algorithm, it is often impossible to directly obtain the minimum $\boldsymbol{u}^*$ of the objective function by the above methods we discussed, so we use proximal operator to define a operator which gives an non-analytical form of the minimum value point. It is defined as following Mathematical expression[15]:

$$\boldsymbol{u}^* = \boldsymbol{prox}_{\boldsymbol{r}}(\boldsymbol{z}) = \mathbf{arg\,min}\left\{\boldsymbol{r}(\boldsymbol{u}) + \frac{\mathbf{1}}{\mathbf{2}}\|\boldsymbol{u}-\boldsymbol{z}\|_2^2\right\} \quad \boldsymbol{u} \in \boldsymbol{R}^{\boldsymbol{n}}$$

Where $\boldsymbol{r}(\boldsymbol{u})$ is the proper, lower-semicontiguous and convex function, and $\boldsymbol{u}^*$ is unique minimum of the right minimization problem.

The proximal operator can also be understood as a mapping process:

We want to find a minimum point $\boldsymbol{u}^*$ which is as close as possible to point $\boldsymbol{z}$ and at the same time the value of this point $\boldsymbol{r}(\boldsymbol{u}^*)$ is as small as possible. Proximal operator is the operator which realize the process we discussed above to map the point $\boldsymbol{z}$ to $\boldsymbol{u}^*$. We can see this process vividly in the diagram below:

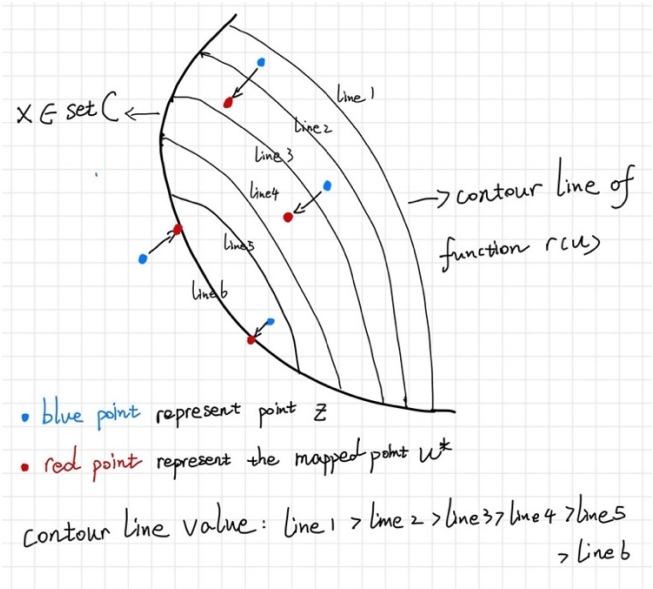


Figure 13: Illustration for proximal operator

It is important to note that the proximal operator only defines an operation and does not give an analytical form solution which highly depends on expression of function $\boldsymbol{r(u)}$. Next we will discuss the analytical form solutions of some functions $\boldsymbol{r(u)}$ that we will use in this chapter.

Here, we give the expressions for the proximal operators of three kinds of r(u) that will be used in the ADMM, and the specific proof process is in the appendix

- indicator function $\boldsymbol{r(u) = l_{C(0,\varepsilon)}(u) = \begin{cases} 0 & \|u\|_2 \le \varepsilon \\ +\infty & otherwise \end{cases}}$

$$\boldsymbol{u^* = prox_{l_{C(0,\varepsilon)}}(z) = min\{\varepsilon, \|z\|_2\} * \frac{z}{\|z\|_2}}$$

- $\boldsymbol{\ell_2}$ norm squared: $\boldsymbol{r(u) = \lambda\|u\|_2^2}$

$$\boldsymbol{u^* = prox_{\lambda\|u\|_2^2}(z) = \frac{z}{2\lambda + 1}}$$

- $\boldsymbol{\ell_1}$ norm: $\boldsymbol{r(u) = \lambda\|u\|_1}$

$$\boldsymbol{u^* = prox_{\lambda\|u\|_1}(z) = \max\{|z| - \lambda, 0\} \cdot sign(z) = shrink(z, \lambda)}$$

Now we know the definition of the proximal operator and the expression for the analytic solutions of the proximal operator for some particular functions $\boldsymbol{r(u)}$.

Let us go back to the question we proposed at the first place. how to solve

$$\boldsymbol{u^* = argmin\{r(u) + h(u)\} \quad u \in R^n}$$

with non-differentiable $\boldsymbol{r(u)}$ and differentiable $\boldsymbol{h(u)}$.

Fortunately, the direct application of proximal operators in convex optimization is the proximal gradient descent method which can be used to answer this question. The iteration steps for proximal gradient descent method show below [16]:

$$\boldsymbol{u_{inter} = u_t - \delta_t \nabla h(u_t)}$$

$$\boldsymbol{u_{t+1} = prox_{r(u)}(u_{inter})}$$

Where $\boldsymbol{\delta_t}$ is the step size which should be smaller than the inverse of Lipschitz constant $\boldsymbol{L}$ of $\boldsymbol{\nabla h}$ in order to make sure convergence:

$$\boldsymbol{\delta_t < \frac{1}{L(\nabla h)}}$$

We can see that proximal gradient descent is made up of two parts. the first part uses an explicit gradient step $\boldsymbol{u_t - \delta_t \nabla h(u_t)}$ to find point $\boldsymbol{u_{\text{inter}}}$ and a second part is implemented by a proximity operator of the non-differentiable regularization term $\boldsymbol{r(u)}$ at the point $\boldsymbol{u_{\text{inter}}}$.

During the first part, because objective function is non-differentiable owing to the non-differentiable regularization term $\boldsymbol{r(u)}$. Therefore, we use the $\boldsymbol{\nabla h(u_t)}$ replacing the gradient of objective function $\boldsymbol{\nabla f(u_t)}$. After that we walk a step size $\boldsymbol{\delta_t}$ along the direction of gradient descent of $\boldsymbol{h(u_t)}$, we find the a 'intermediate point' $\boldsymbol{u_{\text{inter}}}$.

However, Intermediate point $\boldsymbol{u_{\text{inter}}}$ may not be a solution that can make the regularization term $\boldsymbol{r(u)}$ as small as possible because in the first part, we walk an explicit gradient step without considering the $\boldsymbol{r(u)}$.

Therefore, during the second part, we use the proximity operator of the regularization term $\boldsymbol{r(u)}$ at intermediate point $\boldsymbol{u_{inter}}$. The proximity operator can map the $\boldsymbol{u_{inter}}$ to the point $\boldsymbol{u_{t+1}}$ which is as close as possible to point $\boldsymbol{u_{\text{inter}}}$ and at the same time the value of this point $\boldsymbol{r(u)}$ is as small as possible. By iterating through this process, we gradually find the minimum point $\boldsymbol{u^*}$ and this algorithm is proven to converge to a minimum of the original problem[17].

The picture below vividly illustrates this process:

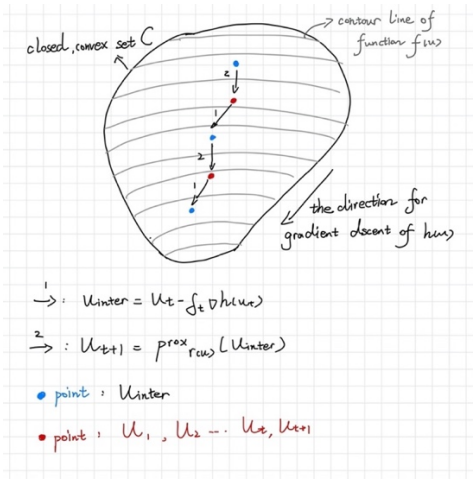


Figure 14: illustration for proximal gradient operator

At this point, our preparation work is complete. This section discusses the definition of proximal operators and the expressions of proximal operators corresponding to different functions $\boldsymbol{r(u)}$. We also discuss a lightweight algorithm called proximal gradient descent. All these preparation work will be applied in the next section of ADMM.

3.2 ADMM for constrained analysis objective function

In this section we will derive the specific form of ADMM for the two objective functions we mentioned earlier.

Recall that we have the constrained analysis objective function below, with convex and non-differentiable regularization term $\boldsymbol{r(x)} = \left\|\boldsymbol{\psi^{\dagger}x}\right\|_{\boldsymbol{1}}$ and convex and non-differentiable data fidelity $\boldsymbol{h(x)} = \boldsymbol{l_{B(0,\varepsilon)}(y - \phi^{\dagger}x)}$.

$$\boldsymbol{x^{*} = arg\,min\left\{\left\|\psi^{\dagger}x\right\|_{1} + l_{B(0,\varepsilon)}(y - \phi^{\dagger}x)\right\}}$$

Because our complex objective function contains two non-differentiable terms, both the gradient descent algorithm and the proximal gradient descent algorithm are unavailable for solving this. Fortunately, we use the Alternating Direction Method of Multipliers (ADMM) algorithm based on Augmented Lagrange Multipliers method to transform this tricky problem.

The *first step* of ADMM to solve this complicated objective function is to do the variable splitting to $\boldsymbol{x}$. The specific method of variable splitting is that we introduce a new variable $\boldsymbol{n}$ and add a linear constraint $\boldsymbol{n = y - \phi^{\dagger} x}$, we can get:

$$\boldsymbol{x = arg\,min\{\ r(x) + h(n)\}\ \ subject\ to\ n = y - \phi^{\dagger} x}$$

For solving this multivariate linear constrained problem in optimization field, Augmented Lagrange Multipliers method is a powerful algorithm.

Therefore, *secondly*, we should transform the function to its Lagrange function:

$$\boldsymbol{\mathcal{L}(x, n, v) = arg\ min\{r(x) + h(n)\ -\ v^{\dagger}(\ n + \phi^{\dagger} x - y)\}.}$$

Thirdly, by introducing the quadratic penalty term, we get the augmented Lagrange function:

$$\boldsymbol{\mathcal{L}(x, n, v) = arg\ min\{r(x) + h(n)\ -\ v^{\dagger}(\ n + \phi^{\dagger} x - y) + \frac{\rho}{2}\left\|n + \phi^{\dagger} x - y\right\|_2^2\}.}$$

Where the $\boldsymbol{v}$ is called the Lagrange multiplier or dual variable and $\boldsymbol{\rho}$ is the penalty parameter for constraint violation.

The *fourth step* is that We define $\boldsymbol{\overline{v} = \rho^{-1} v}$ to scales the augmented Lagrange function by $\boldsymbol{\rho}$. The scale operation to augmented Lagrange function will not influence the value of the minimum point but influence the function value at the minimum point. Then we can get the scaled augmented Lagrange function:

$$\boldsymbol{\mathcal{L}(x, n, v) = arg\ min\{\rho^{-1}\big(r(x_t) + h(n_t)\big) - \overline{v}^{\dagger}\big(\ n + \phi^{\dagger} x - y\big) + \frac{1}{2}\left\|n + \phi^{\dagger} x - y\right\|_2^2\}}$$

The *fifth step* is that we use augmented Lagrange Multipliers method to solve the scaled augmented Lagrange function by the following iteration:

$$(\boldsymbol{x_{t+1}}, \boldsymbol{n_{t+1}}) = \boldsymbol{argmin}\Big\{ \boldsymbol{\rho^{-1}}\big(\boldsymbol{r}(\boldsymbol{x_t}) + \boldsymbol{h}(\boldsymbol{n_t})\big) - \overline{\boldsymbol{v}}_{\boldsymbol{t}}{}^{\dagger}\big(\boldsymbol{n_t} + \boldsymbol{\phi}^{\dagger}\boldsymbol{x_t} - \boldsymbol{y}\big) + \frac{\mathbf{1}}{\mathbf{2}}\big\|\boldsymbol{n_t} + \boldsymbol{\phi}^{\dagger}\boldsymbol{x_t} - \boldsymbol{y}\big\|_{\mathbf{2}}^{\mathbf{2}}\Big\}$$

$$\overline{\boldsymbol{v}}_{\boldsymbol{t+1}} = \overline{\boldsymbol{v}}_{\boldsymbol{t}} - (\boldsymbol{n_{t+1}} + \boldsymbol{\phi}^{\dagger}\boldsymbol{x_{t+1}} - \boldsymbol{y})$$

The dual variable $\overline{\boldsymbol{v}}_{\boldsymbol{t}}$ can be comprehended as follows:

Recall that in the first step, we set the constraint $\boldsymbol{n} = \boldsymbol{y} - \boldsymbol{\phi}^{\dagger}\boldsymbol{x}$. Therefore, we can view the $\boldsymbol{n_t} - (\boldsymbol{y} - \boldsymbol{\phi}^{\dagger}\boldsymbol{x_t})$ as the residual value which represents the violation of the constrain at the t-th iteration. The value of $\overline{\boldsymbol{v}}_{\boldsymbol{t}}$is the sum of the previous $(\boldsymbol{t-1})$th residual values which can be expressed as:

$$-\overline{\boldsymbol{v}}_{\boldsymbol{t}} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{t-1}} ((\boldsymbol{y} - \boldsymbol{\phi}^{\dagger}\boldsymbol{x_i}) - \boldsymbol{n_i})$$

The greater number of violations, the greater value of $-\overline{\boldsymbol{v}}_{\boldsymbol{t}}$, which means that for the solutions $\boldsymbol{x_t}$ which do not satisfy the constraint $\boldsymbol{n_t} + \boldsymbol{\phi}^{\dagger}\boldsymbol{x_t} - \boldsymbol{y} = \mathbf{0}$ at the t-th iteration will increase the augmented Lagrange function value by the term $-\overline{\boldsymbol{v}}_{\boldsymbol{t}}{}^{\dagger}\big(\boldsymbol{n_t} + \boldsymbol{\phi}^{\dagger}\boldsymbol{x_t} - \boldsymbol{y}\big)$. As the number of iterations increases, the value of violation of the constraint increases. In this way, we need to gradually find a solution that satisfies the constraint while minimizing the value of the augmented Lagrange equation.

The *sixth step* is that we use the algorithm Alternating direction method of multipliers (ADMM) to split the joint minimization problem into two sub minimization problems which contain the regularization term $\boldsymbol{r}(\boldsymbol{x_t})$ and data fidelity term $\boldsymbol{h}(\boldsymbol{n_t})$ separately. We can get the following iterations[13]:

$$\begin{cases} \boldsymbol{x_{t+1}} = \boldsymbol{argmin}\Big\{ \boldsymbol{\rho^{-1}} * \boldsymbol{r}(\boldsymbol{x_t}) + \frac{\mathbf{1}}{\mathbf{2}}\big\|\boldsymbol{n_t} + \boldsymbol{\phi}^{\dagger}\boldsymbol{x_t} - \boldsymbol{y} - \overline{\boldsymbol{v}}_{\boldsymbol{t}}\big\|_{\mathbf{2}}^{\mathbf{2}} \Big\} ① \\ \boldsymbol{n_{t+1}} = \boldsymbol{argmin}\{\boldsymbol{\rho^{-1}} * \boldsymbol{h}(\boldsymbol{n_t}) + \frac{\mathbf{1}}{\mathbf{2}}\big\|\boldsymbol{n_t} + \boldsymbol{\phi}^{\dagger}\boldsymbol{x_{t+1}} - \boldsymbol{y} - \overline{\boldsymbol{v}}_{\boldsymbol{t}}\big\|_{\mathbf{2}}^{\mathbf{2}} ② \\ \overline{\boldsymbol{v}}_{\boldsymbol{t+1}} = \overline{\boldsymbol{v}}_{\boldsymbol{t}} - (\boldsymbol{n_{t+1}} + \boldsymbol{\phi}^{\dagger}\boldsymbol{x_{t+1}} - \boldsymbol{y}) \end{cases}$$

For the minimization objective function(①) of $\boldsymbol{x_{t+1}}$,we can see that $\boldsymbol{r}(\boldsymbol{x_t}) = \big\|\boldsymbol{\psi}^{\dagger}\boldsymbol{x}\big\|_{\mathbf{1}}$ is non-differentiable and $\boldsymbol{g}(\boldsymbol{x_t}) = \frac{\mathbf{1}}{\mathbf{2}}\big\|\boldsymbol{n_t} + \boldsymbol{\phi}^{\dagger}\boldsymbol{x_t} - \boldsymbol{y} - \overline{\boldsymbol{v}}_{\boldsymbol{t}}\big\|_{\mathbf{2}}^{\mathbf{2}}$ is differentiable. For this kind of convex optimization problem which contains one non-

differentiable term, we use the proximal gradient descent algorithm which we discussed earlier to solve it. The $\boldsymbol{x_{t+1}}$ has the following expression:

$$\boldsymbol{x_{t+1} = prox_{\rho^{-1}\delta\|\psi^{\dagger}x\|_1}\left(x_t - \delta \cdot \nabla g(x_t)\right)}$$

$$\boldsymbol{= prox_{\rho^{-1}\delta\|\psi^{\dagger}x\|_1}(x_t - \delta \cdot \phi(n_t + \phi^{\dagger}x_t - y - \bar{v}_t))}$$

Where the $\boldsymbol{\delta}$ satisfy the following condition:

$$\boldsymbol{0 < \delta_t < \frac{1}{L(\nabla g)}}$$

$$\boldsymbol{0 < \delta_t < \sigma^{-1}{}_{max}(Real(\phi\phi^{\dagger}))}$$

By leveraging the property of orthogonal transformation (derivation see appendix):

$$\boldsymbol{prox_{\rho^{-1}\delta * r(\psi^{\dagger}x)}(z) = \psi\, prox_{\rho^{-1}\delta * r(x)}(\psi^{\dagger}z)}$$

We can get the $\boldsymbol{x_{t+1} = \psi\, prox_{\rho^{-1}\delta * \|x\|_1}(\psi^{\dagger}(x_t - \delta \cdot \phi(n_t + \phi^{\dagger}x_t - y - \bar{v}_t)))}$

Recall that we have already derived the expression of proximal operator of $\boldsymbol{\ell_1}$ norm in the last section, now we can get the analytical expression for $\boldsymbol{x_{t+1}}$:

$$\boldsymbol{x_{t+1} = \psi\, shink\left(\psi^{\dagger}\left(x_t - \delta \cdot \phi(n_t + \phi^{\dagger}x_t - y - \bar{v}_t)\right), \rho^{-1}\delta\right)}$$

$$\boldsymbol{= \psi \max\left\{\left|\psi^{\dagger}(x_t - \delta \cdot \phi(n_t + \phi^{\dagger}x_t - y - \bar{v}_t))\right| - \rho^{-1}\delta, 0\right\}}$$

$$\boldsymbol{\cdot sign(\psi^{\dagger}(x_t - \delta \cdot \phi(n_t + \phi^{\dagger}x_t - y - \bar{v}_t)))}$$

After that, let us talk about the minimization objective function(②) of $\boldsymbol{n_{t+1}}$, we can find that the this function(②) has the same form as definition of proximal operator. Therefore, the $\boldsymbol{n_{t+1}}$ can be transformed to following expression:

$$\boldsymbol{n_{t+1} = prox_{\rho^{-1} * h(n_t)}(-\phi^{\dagger}x_{t+1} + y + \bar{v}_t)}$$

because $\boldsymbol{h(n_t) = l_{B(0,\varepsilon)}(n_t)}$, we can get the following expression:

$$\boldsymbol{n_{t+1} = prox_{\rho^{-1} * l_{B(0,\varepsilon)}(n_t)}(-\phi^{\dagger}x_{t+1} + y + \bar{v}_t)}$$

Recall that we have already derived the expression of proximal operator of the indicator function in the last section, now we can get the analytical expression for $\boldsymbol{n_{t+1}}$:

$$\boldsymbol{n_{t+1} = P_B(-\phi^{\dagger}x_{t+1} + y + \overline{v}_t)}$$

$$\boldsymbol{= \min\{\varepsilon, \left\|-\phi^{\dagger}x_{t+1} + y + \overline{v}_t\right\|_2\} * \frac{-\phi^{\dagger}x_{t+1} + y + \overline{v}_t}{\|-\phi^{\dagger}x_{t+1} + y + \overline{v}_t\|_2}}$$

In summary, we have the following closed form of ADMM:

$$\begin{cases} \boldsymbol{x_{t+1} = \psi\, shink\left(\psi^{\dagger}\left(x_t - \delta \cdot \phi(n_t + \phi^{\dagger}x_t - y - \overline{v}_t)\right), \rho^{-1}\delta\right)} \\ \boldsymbol{n_{t+1} = P_B(-\phi^{\dagger}x_{t+1} + y + \overline{v}_t)} \\ \boldsymbol{\overline{v}_{t+1} = \overline{v}_t - (n_{t+1} + \phi^{\dagger}x_{t+1} - y)} \end{cases}$$

In order to avoid double-counting the $\boldsymbol{\phi^{\dagger}x_t - y}$ term and improve the iteration speed, we introduce the intermediate variable $\boldsymbol{s_t = \phi^{\dagger}x_t - y}$. Finally, we have the following iteration expression of ADMM algorithm for our constrained analysis objective function:

$$\begin{cases} \begin{cases} \boldsymbol{x_{t+1} = \psi\, shink(\psi^{\dagger}(x_t - \delta \cdot \phi(n_t + s_t - \overline{v}_t)), \rho^{-1}\delta)} \\ \boldsymbol{s_{t+1} = \phi^{\dagger}x_{t+1} - y} \end{cases} \\ \begin{cases} \boldsymbol{n_{t+1} = P_B(-s_{t+1} + \overline{v}_t)} \\ \boldsymbol{\overline{v}_{t+1} = \overline{v}_t - (n_{t+1} + s_{t+1})} \end{cases} \end{cases}$$

3.3 ADMM for unconstrained analysis objective function

Recall that we have the unconstrained analysis objective function below, with convex and non-differentiable regularization term $\boldsymbol{r(x) = \left\|\psi^{\dagger}x\right\|_1}$ and convex and differentiable data fidelity term $\boldsymbol{h(n_t) = \frac{1}{2}\left\|y - \phi^{\dagger}x\right\|_2^2}$.

$$\boldsymbol{x = arg\,min\left\{\lambda\left\|\psi^{\dagger}x\right\|_1 + \frac{1}{2}\left\|y - \phi^{\dagger}x\right\|_2^2\right\} \quad x \in C^N}$$

Compared with the constrained analysis objective function which we discussed in 3.2, we have almost same derivation to get non-analytical ADMM algorithm iterative expression:

$$\begin{cases} \boldsymbol{x_{t+1} = argmin\left\{\lambda\,\rho^{-1} * r(x_t) + \frac{1}{2}\left\|n_t + \phi^{\dagger}x_t - y - \overline{v}_t\right\|_2^2\right\}} \\ \boldsymbol{n_{t+1} = argmin\{\rho^{-1} * h(n_t) + \frac{1}{2}\left\|n_t + \phi^{\dagger}x_{t+1} - y - \overline{v}_t\right\|_2^2} \\ \boldsymbol{\overline{v}_{t+1} = \overline{v}_t - (n_{t+1} + \phi^{\dagger}x_{t+1} - y)} \end{cases}$$

For the solution of $\boldsymbol{x_{t+1}}$, the only tiny difference is that we multiply the regularization parameter $\boldsymbol{\lambda}$ in front of the $\boldsymbol{r(x_t)}$. Because $\boldsymbol{\lambda}$ is a constant, therefore we can directly get the expression for $\boldsymbol{x_{t+1}}$:

$$\boldsymbol{x_{t+1} = \psi\, shink\left(\psi^{\dagger}\left(x_t - \delta \cdot \phi(n_t + \phi^{\dagger}x_t - y - \overline{v}_t)\right), \rho^{-1}\delta\lambda\right)}$$

For solution of $\boldsymbol{n_{t+1}}$, The only difference is that the data fidelity term becomes $\boldsymbol{h(n_t) = \frac{1}{2}\left\|y - \phi^{\dagger}x_t\right\|_2^2}$. Recall that we have already derived the expression of proximal operator of the $\boldsymbol{\ell_2}$ squared norm in the last section, now we can get the analytical expression for $\boldsymbol{n_{t+1}}$:

$$\boldsymbol{n_{t+1} = \frac{-\phi^{\dagger}x_{t+1} + y + \overline{v}_t}{2\rho^{-1} + 1}}$$

For the same reason, we also introduce the intermediate variable $\boldsymbol{s_t = \phi^{\dagger}x_t - y}$. Finally, we have the following iterative expression of ADMM algorithm for our unconstrained analysis objective function:

$$\begin{cases} \begin{cases} \boldsymbol{x_{t+1} = \psi\, shink\left(\psi^{\dagger}\left(x_t - \delta \cdot \phi(n_t + s_t - \overline{v}_t)\right), \rho^{-1}\delta\lambda\right)} \\ \boldsymbol{s_{t+1} = \phi^{\dagger}x_{t+1} - y} \end{cases} \\ \begin{cases} \boldsymbol{n_{t+1} = \frac{\overline{v}_t - s_{t+1}}{2\rho^{-1} + 1}} \\ \boldsymbol{\overline{v}_{t+1} = \overline{v}_t - (n_{t+1} + s_{t+1})} \end{cases} \end{cases}$$

So far, the ADMM algorithm based on convex optimization has been derived for our two objective functions. Next, we will discuss the form of the hybrid algorithm.

3.4 Plug and Play(PnP) hybrid algorithm for constrained analysis objective function

The idea of plug and play algorithm was firstly proposed by Venkata Krishnan et al. by analyzing the MAP (maximum-a-posterior) cost function for ill-posed problems.

They propose that the $\boldsymbol{x = prox_{\rho^{-1}R}(z) = argmin\{R(x) + \frac{\rho}{2}\|x - z\|_2^2\}}$ can be interpreted as the Gaussian denoiser with noise level $\boldsymbol{\sigma = \sqrt{\rho^{-1}}}$ to the noisy picture in the view of MAP[4]:

$$\boldsymbol{x = prox_{\rho^{-1}R(x)}(z) = Denoiser(z, \sqrt{\rho^{-1}})}$$

Now we are going to update the expression in ADMM that we have derived. Recall that we have the following minimization problem of $\boldsymbol{x_{t+1}}$ in ADMM for constrained analysis objective function:

$$\boldsymbol{x_{t+1} = argmin\left\{\rho^{-1} * r(x_t) + \frac{1}{2}\left\|n_t + \phi^{\dagger}x_t - y - \overline{v}_t\right\|_2^2\right\}} \text{①}$$

$$\boldsymbol{x_{t+1} = prox_{\rho^{-1}\delta\|\psi^{\dagger}x\|_1}\left(x_t - \delta \cdot \nabla g(x_t)\right)}$$

$$\boldsymbol{x_{t+1} = Denoiser\left(x_t - \delta \cdot \nabla g(x_t), \sqrt{\delta\rho^{-1}}\right)}$$

Our PnP hybrid algorithm based on ADMM leading to:

$$\begin{cases} \boldsymbol{x_{t+1} = Denoiser\left(\left(x_t - \delta \cdot \phi(n_t + s_t - \overline{v}_t)\right), \sqrt{\delta\rho^{-1}}\right)} \\ \boldsymbol{s_{t+1} = \phi^{\dagger}x_{t+1} - y} \\ \begin{cases} \boldsymbol{n_{t+1} = P_B(-s_{t+1} + \overline{v}_t)} \\ \boldsymbol{\overline{v}_{t+1} = \overline{v}_t - (n_{t+1} + s_{t+1})} \end{cases} \end{cases}$$

$$\boldsymbol{with\ 0 < \delta_t < \sigma^{-1}{}_{max}(Real(\phi\phi^{\dagger}))}$$

3.5 hybrid algorithm based on ADMM for unconstrained analysis objective function

Through the same process, we transform the following minimization problem of $\boldsymbol{x_{t+1}}$ in ADMM for unconstrained analysis objective function:

$$\boldsymbol{x_{t+1} = argmin\left\{\lambda\,\rho^{-1} * r(x_t) + \frac{1}{2}\left\|n_t + \phi^{\dagger}x_t - y - \overline{v}_t\right\|_2^2\right\}}$$

$$\boldsymbol{x_{t+1} = prox_{\rho^{-1}\delta\lambda\|\psi^{\dagger}x\|_1}\left(x_t - \delta \cdot \nabla g(x_t)\right)}$$

$$\boldsymbol{x_{t+1} = Denoiser\left(x_t - \delta \cdot \nabla g(x_t), \sqrt{\delta\rho^{-1}\lambda}\right)}$$

Then, Our PnP hybrid algorithm for unconstrained analysis objective function based on ADMM leading to:

$$\begin{cases} \begin{cases} \boldsymbol{x_{t+1} = Denoiser\left(x_t - \delta \cdot \nabla g(x_t), \sqrt{\delta \rho^{-1} \lambda}\right)} \\ \boldsymbol{s_{t+1} = \phi^{\dagger} x_{t+1} - y} \end{cases} \\ \begin{cases} \boldsymbol{n_{t+1} = \dfrac{\bar{v}_t - s_{t+1}}{2\rho^{-1} + 1}} \\ \boldsymbol{\bar{v}_{t+1} = \bar{v}_t - (n_{t+1} + s_{t+1})} \end{cases} \end{cases}$$

3.6 Pure deep learning method based on U-net convolution neural network

In the inverse imaging problem, we have already discussed the optimization algorithm to reconstruct the original image $\boldsymbol{x}$ from the incomplete sample value y. Another approach is to use the U-net[5] structure to train the convolution neural network which can learn the mapping relationships between the $\tilde{\boldsymbol{x}}$ and $\boldsymbol{x^o}$. By using this method, the reconstruction result $\boldsymbol{x}^*$ can be recovered directly from dirty graph $\tilde{\boldsymbol{x}}$ which is the time domain representation of incomplete sample value $\boldsymbol{y}$.

$$\boldsymbol{x}^* = \boldsymbol{g}(\tilde{\boldsymbol{x}}, \boldsymbol{\Theta})$$

This method is not the focus of this project, but we will compare PnP algorithm with this method to analyze the advantages and disadvantages.

The specific implementation of this method is completed by HaoXiang Sun.

4 Implements

In the chapter 4, we will firstly introduce the implement of ADMM algorithm and PnP hybrid algorithm discussed above which contains the value and estimation of some critical parameters and initialization method. Then, we will introduce the denoising network training section which includes the network structure, network parameters, and network training methods.

4.1 Algorithm implements

4.1.1 ADMM

Now, for solving the constrained analysis objective function:

$$\boldsymbol{x^* = arg\,min\left\{\left\|\psi^\dagger x\right\|_1 + l_{B(0,\varepsilon)}(y - \phi^\dagger x)\right\} \qquad x \in C^N}$$

We have the following iterations:

$$\begin{cases}\begin{cases}x_{t+1} = \psi\ shink\left(\psi^\dagger\left(x_t - \delta \cdot \phi(n_t + s_t - \overline{v}_t)\right), \rho^{-1}\delta\right)\\ s_{t+1} = \phi^\dagger x_{t+1} - y\end{cases}\\ \begin{cases}n_{t+1} = P_B(-s_{t+1})\\ \overline{v}_{t+1} = \overline{v}_t - (n_{t+1} + s_{t+1})\end{cases}\end{cases}$$

For solving unconstrained analysis objective function:

$$\boldsymbol{x = arg\,min\left\{\lambda\left\|\psi^\dagger x\right\|_1 + \frac{1}{2}\left\|y - \phi^\dagger x\right\|_2^2\right\} \quad x \in C^N}$$

We have the following iterations:

$$\begin{cases}\begin{cases}x_{t+1} = \psi\ shink\left(\psi^\dagger\left(x_t - \delta \cdot \phi(n_t + s_t - \overline{v}_t)\right), \rho^{-1}\delta\lambda\right)\\ s_{t+1} = \phi^\dagger x_{t+1} - y\end{cases}\\ \begin{cases}n_{t+1} = \dfrac{\overline{v}_t - s_{t+1}}{2\rho^{-1} + 1}\\ \overline{v}_{t+1} = \overline{v}_t - (n_{t+1} + s_{t+1})\end{cases}\end{cases}$$

It is good to have a definite form to solve our problems.However, that is not enough to start our algorithm. In the process of implementation, we still need to estimate and consider some parameters and we also need to determine the method of initialization and the conditions to terminate the iterations.

4.1.1.1 Generate pending data

We already know that we are going to solve the problem of recovering the original $\boldsymbol{x}$ from the incomplete samples $\boldsymbol{y}$ in the frequency domain. Therefore, at the beginning of the project we're going to simulate the subsampling process and generate the data $\boldsymbol{y}$ that we actually get in the real problem. According to the linear sensing model we discussed earlier, y has following expression:

$$\boldsymbol{y} = \boldsymbol{\phi}^{\dagger}\boldsymbol{x}^{\boldsymbol{o}} + \boldsymbol{n_1}$$

In the project, the sparse basis matrixes $\boldsymbol{\psi}$ $\boldsymbol{\psi}^{\dagger}$ and measurement matrixes $\boldsymbol{\phi}$ $\boldsymbol{\phi}^{\dagger}$ are given in the form of forward and inverse anonymous functions which we can invoke directly.

The noise $\boldsymbol{n_1}$ is additive white gaussian noise with standard deviation $\boldsymbol{\sigma_1}$. To keep the input RSNR value around 30, we set the sigma value at $\boldsymbol{\sigma_1 = 0.07}$. It is important to note that $\boldsymbol{n_1}$ here is noise in the frequency domain, which is not the same as the time domain noise $\boldsymbol{n_2}$ which is directly added to the image, which we will introduce in the next section of image denoising.

In this way, we can get the pending data $\boldsymbol{y}$.

4.1.1.2 Estimated free parameter value

In the project, we have already been given some off-the-shelf estimation function for some parameters, such as noise boundary $\boldsymbol{\varepsilon = \sigma_1\sqrt{m + \sqrt{m}}}$ and regularization parameter $\boldsymbol{\lambda = (1e-3) * norm(alphad, inf)}$ ($\boldsymbol{alphad}$ is the wavelet decomposition vector of dirty graph $\tilde{\boldsymbol{x}}$). Now there are only two critical parameters that need to be carefully considered, the penalty parameter $\boldsymbol{\rho}$ and the step size $\boldsymbol{\delta_t}$we are, we're going to talk about the value of $\boldsymbol{\rho}$ and leave $\boldsymbol{\delta_t}$ in the convergence analysis.

In ADMM algorithm, the $\boldsymbol{\rho}$ is the free parameter which will usually be fine-tuned to maximize the reconstruction result. However, there is no recommended range for its

value. We will try several $\boldsymbol{\rho}$ values and draw the figure between reconstruction results and rho $\boldsymbol{\rho}$ values.

4.1.1.3 Convergence analysis

In order to make sure convergence of algorithm, the step size should be smaller than the inverse of Lipschitz constant $\boldsymbol{L}$ of $\boldsymbol{\nabla g(x_t)}$. For $\boldsymbol{g(x_t) = \frac{1}{2}\left\|n_t + \phi^{\dagger} x_t - y - \bar{v}_t\right\|_2^2}$, the upper boundary for $\boldsymbol{\delta_t}$ has the following expression:

$$\boldsymbol{\delta_t \leq \frac{1}{L(\nabla g)} = \frac{1}{\sigma_{max}^2(Real(\phi^{\dagger}))}}$$

Where $\boldsymbol{\sigma_{max}}$ represents the maximum eigenvalue of $\boldsymbol{Real(\phi\phi^{\dagger})}$.We use MATLAB code

```
%convergence check
  uni_matrix = eye(Nx);
  [U,S,V] = svd(Phi_t(uni_matrix));
  L = max(max(S*S'));
```

Figure 15: MATLAB code for calculate Lipschitz constant $\boldsymbol{L}$

We can calculate this upper boundary:

$$\boldsymbol{\delta_t \leq 2.019 \times 10^{-7}}$$

Howeverdoes notdoesn't mean that any value less than this upper bound is available, because if the step size is too small, it will manya lot of iterations get to the optimal solution. Empirically, by combing the reconstruction speed and quality, we find the proper lower bound has:

$$\boldsymbol{1 \times 10^{-8} \leq \delta_t}$$

In summary, we set $\boldsymbol{\delta_t}$ should has the following range:

$$\boldsymbol{1 \times 10^{-8} \leq \delta \leq 2.019 \times 10^{-7}}$$

4.1.1.4 Variable initialization

Because of the robustness of our ADMM algorithm, the final results of the algorithm are not sensitive to the initial value. But a suitable initialization method can indeed

help algorithm converge to the optimal results as soon as possible. Here, in the iterative expression of ADMM, we initialize four variables $\boldsymbol{x_1}\ \boldsymbol{v_1}\ \boldsymbol{s_1}\ \boldsymbol{n_1}$which participate in the iteration. Instead of initializing $\boldsymbol{x_1}$ from the dirty image $\boldsymbol{x_1} = \tilde{\boldsymbol{x}}$,here, we firstly initialize $\boldsymbol{x_1} = \boldsymbol{zeros(Nx, Ny)}$ and substitute this value to the expression to calculate other variables:

$$\begin{cases} \boldsymbol{x_1} = \boldsymbol{zeros(Nx, Ny)} \\ \boldsymbol{s_1} = -\boldsymbol{y} \end{cases}$$
$$\begin{cases} \boldsymbol{n_1} = \boldsymbol{P_B}(-\boldsymbol{s_1}) \\ \bar{\boldsymbol{v}}_1 = -(\boldsymbol{n_1} + \boldsymbol{s_1}) \end{cases}$$

The advantages of this initialization method are that on the one hand, because we do not know how big the difference between the dirty image $\tilde{\boldsymbol{x}}$ and the original graph $\boldsymbol{x^o}$is. If there is a big gap between them, then the initialization value of the dirty graph may start from an inferior value (see figure 17 below) and takes many iterations to be normal. On the other hand, recall what we talked about in the previous chapter. The larger the value of $\boldsymbol{v}$, the more it forces the value of $\boldsymbol{x_t}$ to satisfy the constrains. Now we already had a larger value of $\boldsymbol{v}$ by using the initialization method above. This advantage, however, is not provided by another method which initializes all variables to zero.

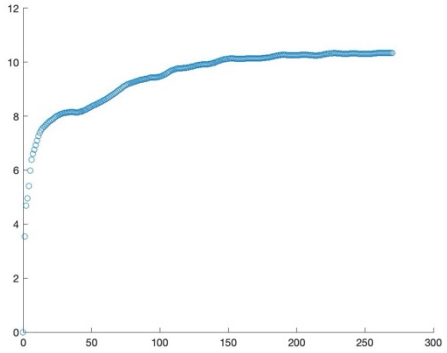


Figure 16 : reconstruction process of initialization that we proposed

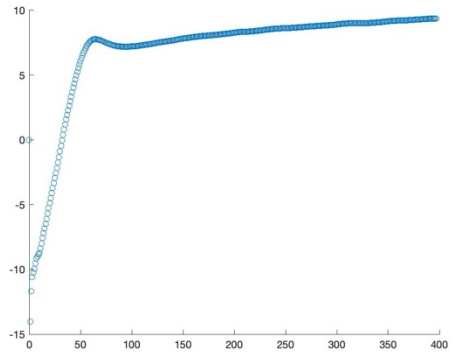


Figure 17: reconstruction process for initialization at dirty figure $\boldsymbol{x_1 = \tilde{x}}$

4.1.1.5 Termination conditions

After a certain number of iterations, the value of the solution converges and tends to be stable which is nothing need to continue. therefore, we should set the conditions to end the iteration. we define the following constant:

$\boldsymbol{rel\_tol1}$: represent the minimum relative change of the objective function value. (default:1e-5)

$\boldsymbol{rel\_tol2}$: represent the minimum relative change of the iterations. (default:1e-5)

$\boldsymbol{max\ \_iter}$: represent the maximum iterations. (default:500)

the algorithm can be stopped by meeting any of the following conditions:

$$\begin{cases} \dfrac{|f(x_t) - f(x_{t-1})|}{f(x_t)} < rel\_tol1 \\ \dfrac{\|x_t - x_{t-1}\|_2}{\|x_t\|_2} < rel\_tol2 \\ t > max\ \_iter \end{cases}$$

4.1.2 PnP (Plug and Play) hybrid algorithm

Inspired by the idea of MAP, we use the denoiser operator to replace the proximal operator of regularization term to build our PnP algorithm. We have the following expression:

For the constrained analysis objective function:

$$x_{t+1} = prox_{\rho^{-1}\delta\|\psi^\dagger x\|_1}\left(x_t - \delta \cdot \nabla g(x_t)\right) \rightarrow x_{t+1} = Denoiser\left(\left(x_t - \delta \cdot \nabla g(x_t)\right), \sqrt{\delta\rho^{-1}}\right)$$

for the unconstrained analysis objective function:

$$\boldsymbol{x_{t+1} = prox_{\rho^{-1}\delta\lambda\|\psi^{\dagger}x\|_1}\left(x_t - \delta \cdot \nabla g(x_t)\right) \rightarrow x_{t+1} = Denoiser\left(x_t - \delta \cdot \nabla g(x_t), \sqrt{\delta\rho^{-1}\lambda}\right)}$$

Because PnP hybrid algorithm is constructed on the bases of ADMM algorithm which means that they have almost initialization methods and termination conditions. Again, we will talk about step size in the convergence analysis. Besides, we also need to determine the noise level where the denoiser operate on.

4.1.2.1 Convergence analysis

Because the denoiser is non-convex and non-differentiable, the convergence of the PnP hybrid algorithm becomes complicated. At present, most of the convergence analysis of the hybrid algorithm remain in the empirical proof of convergence, that is, the authors observe the convergence results. The few analytical papers remain on proving that the hybrid algorithm can converge at fixed points[18] [19]. It is worth noting that fixed points may not be optimal points. This is one of the shortcomings of PnP algorithm which leave the space for development of robust algorithm.

Here, after a large number of tests, we found that for the hybrid algorithm, a too large value of step size $\boldsymbol{\delta}$ would lead to unconvergence of the reconstruction results, and a too small value would not only affect the iteration speed, but also lead to poor iteration results. So, we finally select $\boldsymbol{\delta = 1e-7}$ and $\boldsymbol{\delta = 2e-7}$ two values.

4.1.2.2 Denoising level and Denoiser

Here, we are going to determine the noise level where the denoiser operate on.

For the constrained analysis objective function:

$$\begin{cases} \boldsymbol{x_{t+1} = Denoiser\left(\left(x_t - \delta \cdot \phi(n_t + s_t - \bar{v}_t)\right), \sqrt{\delta\rho^{-1}}\right)} \\ \boldsymbol{s_{t+1} = \phi^{\dagger}x_{t+1} - y} \\ \begin{cases} \boldsymbol{n_{t+1} = P_B(-s_{t+1} + \bar{v}_t)} \\ \boldsymbol{\bar{v}_{t+1} = \bar{v}_t - (n_{t+1} + s_{t+1})} \end{cases} \end{cases}$$

We can find that the denoising level $\boldsymbol{\sigma_2 = \sqrt{\delta\rho^{-1}}}$. However, we note that the free parameter $\boldsymbol{\rho}$, which determines the denoising level, does not actually participate in the iterative loop of the algorithm. This means that no matter what the value of the $\boldsymbol{\delta}$ is, we can get the desired denoising level $\boldsymbol{\sigma_2}$ by freely changing the value of the $\boldsymbol{\rho}$. In other words, the denoising level $\boldsymbol{\sigma_2}$ here is also a free parameter. We will fine-tune the $\boldsymbol{\sigma_2}$ of the range $[\mathbf{0.01}, \mathbf{0.16}]$ to get the best reconstruction.

For the unconstrained analysis objective function:

$$\begin{cases} \begin{cases} \boldsymbol{x_{t+1} = Denoiser\left(x_t - \delta \cdot \nabla g(x_t), \sqrt{\delta\rho^{-1}\lambda}\right)} \\ \boldsymbol{s_{t+1} = \phi^{\dagger} x_{t+1} - y} \end{cases} \\ \begin{cases} \boldsymbol{n_{t+1} = \dfrac{\overline{v}_t - s_{t+1}}{2\rho^{-1} + 1}} \\ \boldsymbol{\overline{v}_{t+1} = \overline{v}_t - (n_{t+1} + s_{t+1})} \end{cases} \end{cases}$$

We can see the denoising level $\boldsymbol{\sigma_2 = \sqrt{\delta\rho^{-1}\lambda}}$ where parameter $\boldsymbol{\lambda}$ is estimated by the function $\boldsymbol{\lambda = (1e-3) * norm(alphad, inf)}$. Furthermore, parameter both $\boldsymbol{\delta}$ and $\boldsymbol{\rho}$ are involved in the iteration of the algorithm. This means that the denoising level value $\boldsymbol{\sigma_2}$ will entirely depend on the value of these parameters which are used in the algorithm. As discussed in the ADMM algorithm, for the free parameter rho $\boldsymbol{\rho}$, we usually need constantly fine-tuning to maximize the quality of the reconstruction. Due to the relationship above, as the $\boldsymbol{\rho}$ adjust continually, our expected denoising levels $\boldsymbol{\sigma_2}$ also change synchronously. However, our denoisers usually are trained neural networks, which means that this tuning process increases the amount of computation dramatically and makes the $\boldsymbol{\rho}$ a problematic parameter to adjust. To avoid this, instead of adjusting the $\boldsymbol{\rho}$, we adjust the noise level of the range $[\mathbf{0.01}, \mathbf{0.16}]$. When we test the reconstruction result of the different noise levels in the PnP hybrid algorithm, we calculate and use the $\boldsymbol{\rho}$ value at corresponding denoising level.

For the denoiser, in this project, we will train and use the de-noising neural network. In order to reflect the powerful ability of the de-noising neural network, we will also

use the current advanced hand-designed de-noising algorithm BM3D to make a comparison with it. Next, we will discuss how to train a neural network.

4.2 Denoising Neural Network

The purpose of image denoising is to estimate the potential clean image x from its noise observation value $\boldsymbol{x_{noisy}}$. For image denoising, its degradation model hasthe following expression:

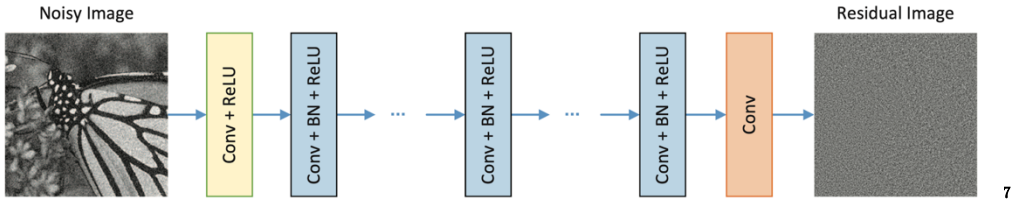
$$\boldsymbol{x_{noisy} = x + n_2}$$

Where we assume $\boldsymbol{n_2}$ is the additive white Gaussian noise with noise energy $\boldsymbol{{\sigma_2}^2}$.In the past few decades, some prior-based denoising approaches such as BM3D which is based on the nonlocal self-similarity are prevalent. However, these methods suffer from the disadvantages of time-consuming and free parameter-tuning. Recent years, benefiting from the improvement of data science, the convolutional neural network shows promising ability in image recovery such as image denoising. In this section, we are going to implement the denoiser by training the denoising neural network.

4.2.1 Network structure

Firstly, we will introduce the network structure, which is called DnCNN-S. The DnCNN-S proposed by Zhang Kai et al.[20] is a very powerful denoising network. It

has the following architecture:



[7]

Figure 18: Structure for DNCNN-S

The overall architecture has 17 layers; the specific parameters for each layer are given in the following table:

layer	layer structure	number of filters	filter size
1	Convolution layer+ Relu layer	32	3*3*1
2~16	Convolution layer+ Batch normalization layer+Relu layer	32	3*3*32
17	Convolution layer	1	3*3*32

The reason that we use the DnCNN-S structure is because it has the following advantages:

Firstly, compared with the other denoising models such as MLP[21], which try to recover latent image $\boldsymbol{x}$ from the noisy image $\boldsymbol{x_{noisy}}$. The authors here apply residual learning which tries to predict noise $\boldsymbol{n}$ from the noisy image $\boldsymbol{x_{noisy}}$. The reason why the residual mapping $\boldsymbol{g}(\boldsymbol{x_{noisy}}) = \boldsymbol{n_2}$ is better than original mapping $\boldsymbol{g}(\boldsymbol{x_{noisy}}) = \boldsymbol{x}$ for denoising is that Neural networks are not good at identical mapping. If the layers behind the deep network are identical mappings, the deep network degenerates into a shallow network. The original mapping is more like an identical mapping than residual mapping, especially under the low denoising level. There is also another advantage for DnCNN-S. We noticed that each convolution layer, followed by a batch normalization. For a neural network which contains many hidden layers, the input distribution of hidden layers changes during training due to

[7] *image stems from the paper: Zhang, K., et al., Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising. IEEE Transactions on Image Processing, 2017. 26(7): p. 3142-3155.*

the change of parameters in each layer, which makes the network model difficult to be stable. We call this problem internal covariate shift. By introducing batch normalization, the distribution of input values for neurons in each layer is forcibly pulled back to a standard normal distribution with a mean of 0, and a variance of 1 which reduces internal covariate shift effectively.

Besides, the authors prove that by combining the batch normalization and residual learning together, we can accelerate the training process and improve the results[21]

4.2.2 Dataset

In order to generate the training and validation set, we firstly improve the data set we received in this project. We found that there are a small number of all-black images in this data set, and there are also few images which contain a large number of black areas. we decided to eliminate these two kinds of pictures owing to the following reasons:

(1) From the view of network training, the whole black image $\boldsymbol{x}$ is a zero matrix. In other words, the noisy image $\boldsymbol{y}$ will be equal to noise $\boldsymbol{n_2}$. When we feed this kind of image to DnCNN-S for residual learning $\boldsymbol{g}(\boldsymbol{y}) = \boldsymbol{n_2}$, the identity mapping occurs, which leads to the degradation.

(2) From the view of the image. The whole black image is actually not the image because it does not contain the texture structure and information.

Figure 20: all black image

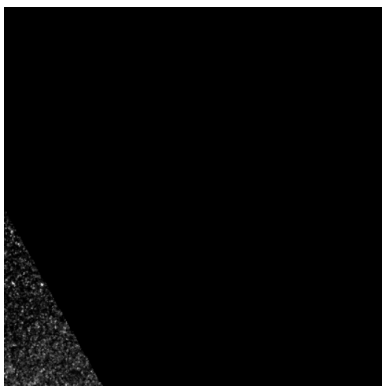


Figure 19: image with a large number of black areas

Following that, we randomly selected 1% of the images from the total data set as the validation set and use the remaining 99% as the training set. We use data structure denosingimagedatastore, which is specifically designed to generate (noisy image, noise) pairs to use in residual denoising neural network. In addition, we will randomly grab image patches to generate the training and validation set of sizes (64,128,256,512) to train and analyze the performance of networks under different input size.

4.2.3 Network parameters and hyperparameters

4.2.3.1 parameters initialization

In the neural network, parameters refer to the variables obtained by the model through learning, such as weights and offset in the network layer.

The initialization of parameters in the network layer mainly focuses on the convolution layer. Most of the parameter in the convolution layer can use the default value except the weight. The default initialization method of weight is zero initialization, which will lead to network degradation. The reasons are as follows：

In the neural network, we know that the number of neurons in the layer is the number of data features. In the hidden layer, the number of neurons also represents the number of new "extracted" data features after data processing. The more neurons in a hidden layer, the more new features extracted in this layer.

In general, initializing all the weights to zero results in the network failing to break the symmetry. This means that every neuron in the same layer learns the same things which are no different from using only one neuron in the layer.

```
conv = convolution2dLayer(3, numfilters, ...
    'Stride', 1, ...
    'Padding', 'same', ...
    'WeightLearnRateFactor', 1, ...
    'WeightL2Factor', 1, ...
    'BiasLearnRateFactor', 1, ...
    'BiasL2Factor', 0, ...
    'Name', 'Conv1');
conv.Weights = sqrt(2/(9*numfilters))*randn(3,3,channels,numfilters,'single');
conv.Bias = zeros(1,1,numfilters,'single');
```

Figure 21: weight initialization method

Therefore, we use the He-initialization[22] method to randomly generate weight each layer which not only can avoid degradation problem we talk above but also it can prevent the network from gradient explosion or gradient disappearance[23].

4.2.3.2 Hyperparameter parameter adjustment

In the neural network, Hyperparameter refers to the parameter set according to experience, which cannot be obtained through learning and can affect the value of weight and bias, such as the number of iterations, the number of hidden layers, the number of neurons in each layer, initial learning rate, etc.

Due to the limited computing power and time, we will only focus on the three hyperparameter, which are initial learning rate, learning rate drop period and epochs. We choose these three parameters not only because they have a higher priority in the parameters, but also because these three super parameters determine the 'amount' of learning. For the remaining super parameters, we will choose the default value. We give the adjustment range in the following table:

Hyperparameter	Adjustment range
Epochs	6~20
learning rate drop period (drop factor 0.1)	3~10
initial learning rate	1e-2 or 1e-3

Next, we will discuss the hyperparameter adjustment method we used in this project. Here, we impose the Bayesian optimizer on our training program to find the optimal hyperparameter. The Bayesian optimizer is based on the Bayesian optimization algorithm, which is a very suitable algorithm to optimize the hyperparameter of the regression model in deep learning. The specific optimization process is the following:

We deliver the range of hyperparameter which need to be optimized to the optimizer. The optimizer will firstly try several combinations to train networks and use these denoising networks to calculate the objective function value. Then the optimizer predicts the location of the next point, which represent the hyperparameter combination. The denoising network trained by this predicted combination may reduce the value of the objective function with a high probability. Through continuous attempts, the optimizer will build a hyperparameter model and finally reach the optimal combination of the hyperparameter, which has the minimum objective function value. The following figure represents a two-dimensional Bayesian optimizer whose optimization parameters are learning rate drop period and Epochs number. The objective function for this optimizer is

$$\boldsymbol{obj = -1 * denoising\ result}$$

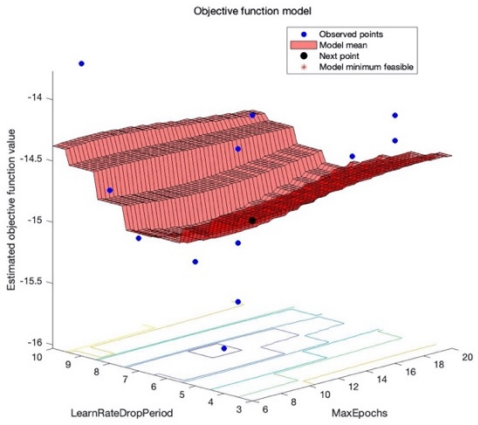


Figure 22: Objective function model generated by the optimizer

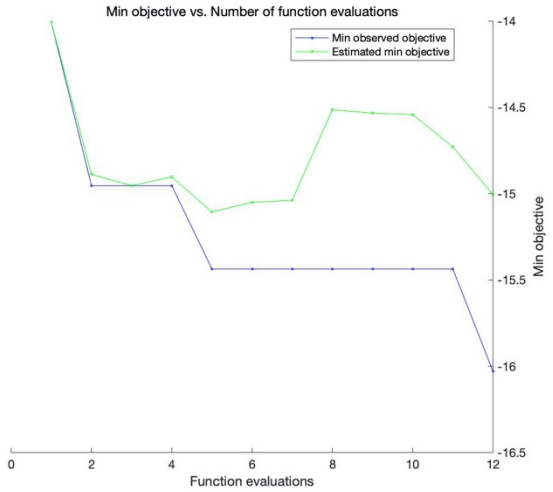


Figure 23: the minimum value of objective function vs the number of evaluations

We can see that through continuous optimization, the objective function decreases from -14 to -16, which means that our denoising effect has increased from 14 to 16.

In fact, what we are actually optimizing is that a three-dimensional Bayesian optimizer contains the three hyperparameters we discussed above in the table. However, because the network training program, including the three-dimensional Bayesian optimizer needs to run on the remote GPU which neither has a client nor plotting a graph. Therefore, we did not have the diagram which can explicitly describe the optimization process, but we will put our code in the appendix.

5 Result and improvements

5.1 Methodology for evaluation

In this chapter, we will analyze our results. In order to judge the quality of the results by a unified standard, we will analyze the following criteria：

- RSNR(reconstruction signal to noise ratio)

$$\mathbf{RSNR} = \mathbf{20log_{10}}\left(\frac{\|\boldsymbol{x^o}\|_2}{\|\boldsymbol{x^o} - \boldsymbol{x^*}\|_2}\right)$$

RSNR is a standard in the data point of view. It is used to evaluate the intensity of ground-truth image $\|\boldsymbol{x^o}\|_2$ to intensity of reconstruction error $\|\boldsymbol{x^o} - \boldsymbol{x^*}\|_2$. The larger this value is, the smaller the reconstruction error is relative to the ground-truth image.

- SSIM (structural similarity index)

SSIM is a standard in the image point of view. It is used to measure the similarity between the ground-truth image $\boldsymbol{x^o}$ and reconstruction image $\boldsymbol{x^*}$. The higher the SSIM value, the closer the reconstructed image is to the original, which means the better the reconstructed image is visual.

- Time

We are going to analyze the time that each reconstruction algorithm takes.

In addition to the unified evaluation criteria, we will use 24 randomly selected images from the data set to form a unified test set (the information of the test set will be shown in the appendix). We use our algorithm to reconstruct these 24 images, respectively.

Based on the above criteria and test set, we will conduct a comprehensive analysis to the results of the ADMM algorithm, PnP hybrid algorithm and Pure Neural Network method. At the same time, by analyzing the results, we will answer the following questions:

(1) The relationship between the reconstruction results of ADMM and free parameter rho
(2) Which factors of denoising network will affect the reconstruction result of PnP algorithm? Here we mainly focus on the patch size and denoising level.
(3) Comparison between the ADMM algorithm and PnP algorithm and pure neural network method. We will discuss the advantages and disadvantages of each algorithm.

5.2 ADMM

We are going to present the data for all of them. However, due to the layout and pages constraints of paper, we will only show 4 out of the 24 images corresponding to the best and worst SNR value, best and worst SSIM value.

5.2.1 ADMM reconstruction result

For ADMM of the constrained objective function, its reconstruction results of the test set are as follows. The values of rho and delta are the values of free parameters used in the reconstruction of current data.

figure	1	2	3	4	5	6	7	8	9	10	11	12
rsnr	8.0458	5.7741	4.1581	5.1378	5.9857	6.8365	4.6459	9.2008	2.993	4.3895	5.6283	11.317
ssim	0.46093	0.58842	0.47574	0.48279	0.4357	0.60845	0.4267	0.6359	0.63051	0.48892	0.49418	0.8332
time(sec)	25.242	52.551	17.476	25.852	10.751	52.452	24.242	17.12	31.61	7.5324	56.444	28.747
figure	13	14	15	16	17	18	19	20	21	22	23	24
rsnr	5.1255	5.2397	4.3588	3.2193	3.6603	5.3344	4.2308	3.7084	12.039	4.0115	7.5996	6.6027
ssim	0.49259	0.56622	0.70397	0.43945	0.47222	0.60257	0.49263	0.46415	0.52737	0.62322	0.46888	0.50053
time(sec)	1.8831	26.166	22.836	7.9755	13.258	5.2495	12.509	2.7059	7.4264	20.25	68.728	36.373
average rsnr:	5.8018		rho :	1								
average ssim:	0.53813		delta:	1.00E-07								
average time:	23.97415833											

Table 1: Reconstruction result for the constrained objective function

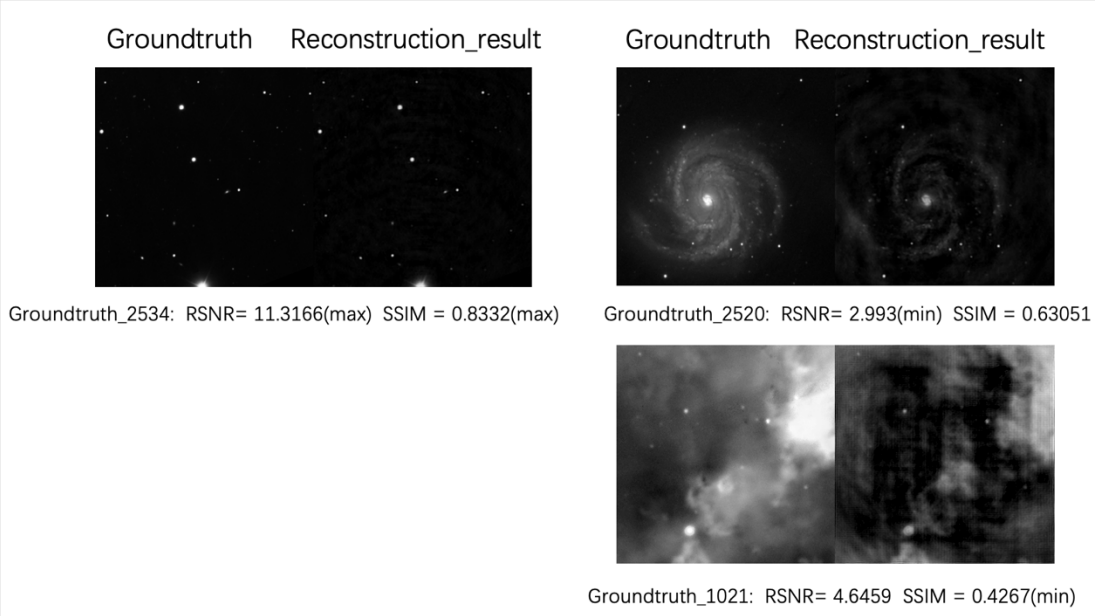


Figure 24: Reconstruction image for the constrained objective function

For ADMM of the unconstrained objective function, its reconstruction results of the test set are as follows.

figure	1	2	3	4	5	6	7	8	9	10	11	12
rsnr	7.8149	5.2823	5.0104	4.9629	6.2065	7.075	8.2607	10.891	4.1613	4.7842	5.5185	11.912
ssim	0.46056	0.66753	0.60013	0.43981	0.5447	0.6678	0.6008	0.7568	0.73	0.5598	0.6409	0.8843
time(sec)	108.78	109.57	105.98	43.544	36.015	109.06	109.08	109.26	98.268	109.95	109.45	31.457
figure	13	14	15	16	17	18	19	20	21	22	23	24
rsnr	6.2965	8.5172	5.4064	3.2927	4.1617	6.5689	4.9156	4.7843	12.058	5.0356	7.7807	8.5044
ssim	0.60429	0.72659	0.75841	0.49683	0.5232	0.7369	0.5819	0.597	0.5492	0.6866	0.6679	0.6176
time(sec)	109.85	109.36	110.13	109.83	108.77	109.1	67.798	33.9	109.85	109.39	109.24	109.53
average rsnr:	6.6334		rho :	1								
average ssim:	0.62914		delta:	1.00E-07								
average time:	94.88175											

Table 2: reconstruction result for unconstrained objective function

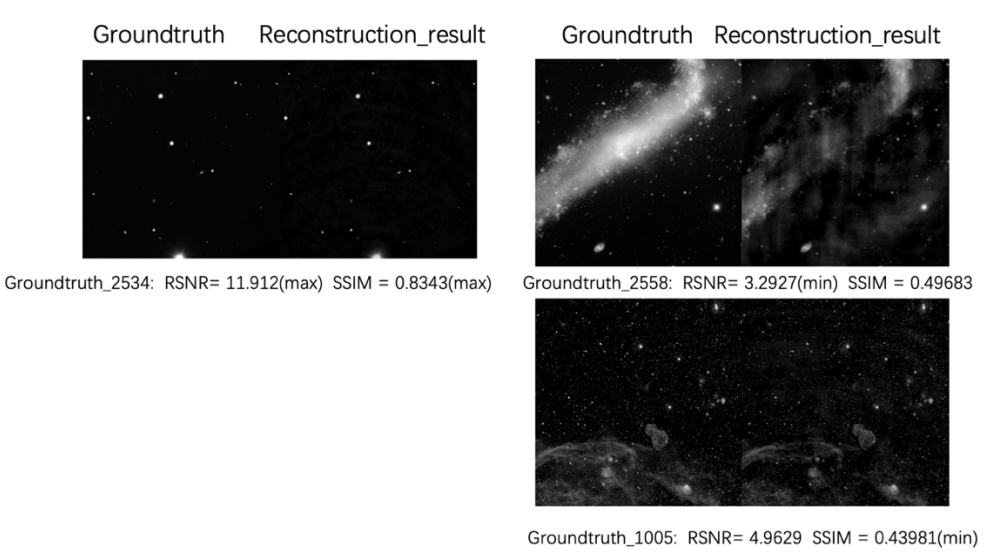


Figure 25: Reconstruction image for the unconstrained objective function

We can see that no matter for the constrained objective function or unconstrained objective function, ADMM algorithm can achieve better reconstruction results for images with only a few stars. However, for the image of the milky way and Nebula, which contains complex texture structure and hierarchical sense, the reconstruction effect is inferior.

5.2.2 ADMM reconstruction result with free parameter rho $\boldsymbol{\rho}$

Now we will assign the value of $\boldsymbol{\rho}$ in the form of exponentially increasing(0.01 0.1 1 10 100 1000), so as to quickly explore the value of Rho which can maximize the reconstruction.

For ADMM of constrained objective function:

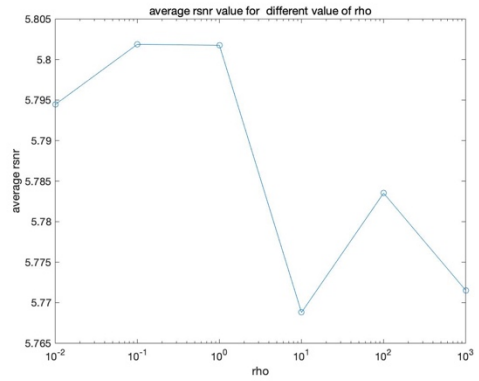


Figure 26: Reconstruction RSNR vs Rho value for the constrained objective function

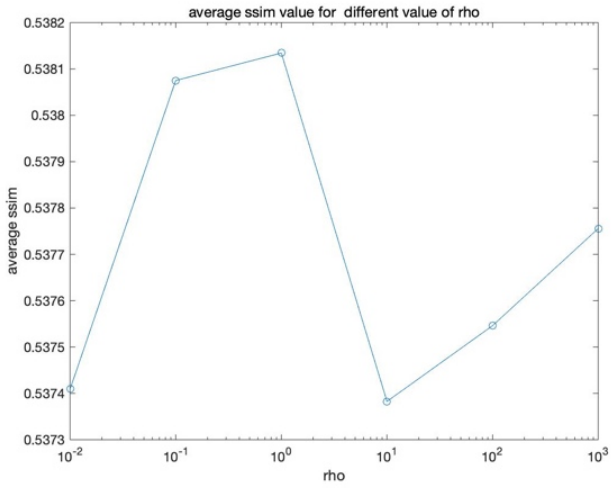


Figure 27: Reconstruction SSIM vs Rho value for the constrained objective function

We can see that even if we assign the value of ρ in the form of exponential increasing, The value of objective function only fluctuates slightly, which shows that the value of ρ has little influence on the constrained objective function.

For ADMM of unconstrained objective function:

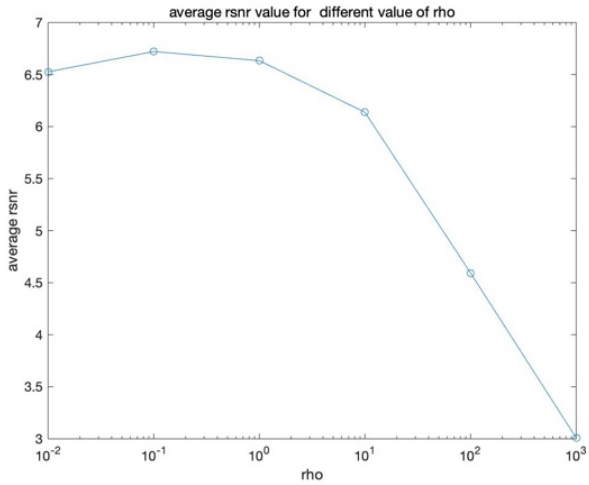


Figure 28: Reconstruction RSNR vs Rho value for the unconstrained objective function

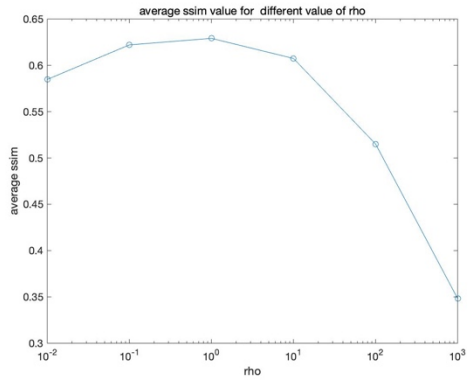


Figure 29:Reconstruction SSIM vs Rho value for the constrained objective function

We can see that for the unconstrained objective function, a proper ρ value should be taken in the range of [0.1,10]. We find that when the rho is greater than 10, the reconstruction result of the objective function will become worse.

5.3 PnP hybrid algorithm

5.3.1 Patch size

When we first trained the denoising network for PnP algorithm, we randomly grabbed patches with size 32*32 from the training set to train our input networks. Our original intention was to make the neural network easier to train by reducing the size of the patches, so as to achieve better denoising and reconstruction effect. However, in our practice, we found that no matter how to optimize the parameters through the Bayesian optimizer, the results of our network denoising and reconstruction will have obvious boundary effect (see figure below). This forced us to change our thinking to train a larger input size denoising neural network.

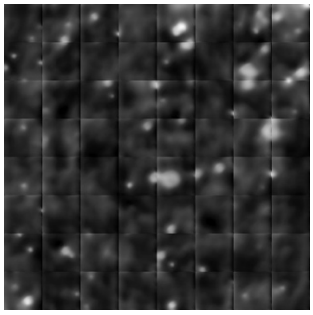


Figure 30: reconstruction image with obvious boundary effect

Under the condition of fixed denoising level of 0.07, we successively increased the input size of the network from 32 to 64 to 128 to 256 to 512. We selected the best network of the same size to test and reconstruct. Finally, we plot the three variables of network size, denoising effect and reconstruction effect in the figure below. The specific results of the different sizes of the network will be included in the appendix.

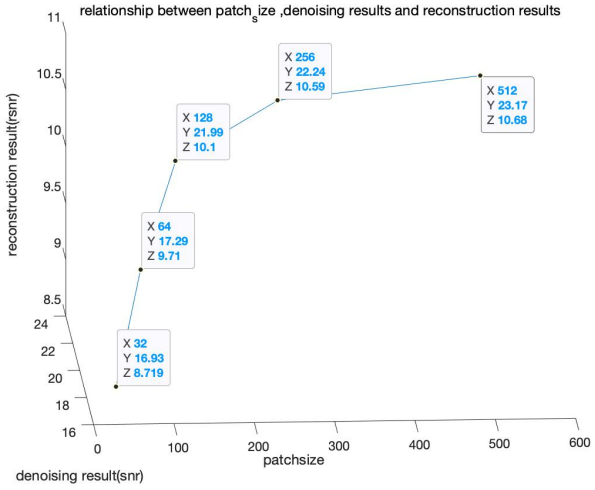


Figure 31: Reconstruction RSNR vs Rho value for the constrained objective function

We were surprised to find that as the input size of the network increased, both the denoising and reconstruction effects increased.

We analyzed the following two reasons. On the one hand, although the network with small input size is easy to train, the convolution kernel is unable to extract information from the context at the edge, resulting in boundary effect, which will reduce the denoising effect. On the other hand, when applying de-noising network in the reconstruction process of hybrid algorithm, because the cutting position of patch block is fixed, compared with the centre of patch block, the edge of patch block is always not reconstructed well. As the number of iterations increases, the boundary effect during reconstruction becomes more and more obvious.

5.3.2 Denoising level

Now, we know that the best network input size is 512. Then, for our denoising network at what denoising level will make our reconstruction results the best?

Here, we train the denoising network with the input size of 512 from 0.01 to 0.16. For these denoising networks, the denoising effect at their respective noise levels is similar (The values fluctuate around 22.5). We plot the two variables of denoising level and reconstruction effect in the figure below. The specific results of the different denoising levels of the network will be included in the appendix.

For PnP algorithm of constrained objective function:

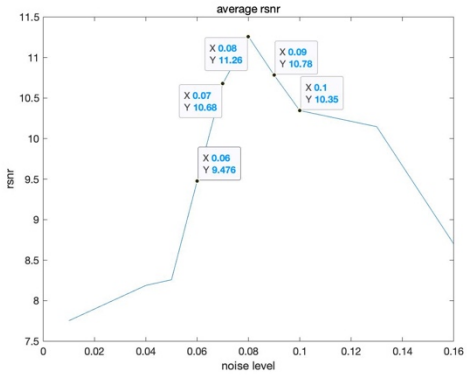


Figure 33: Reconstruction average RSNR vs denoising level

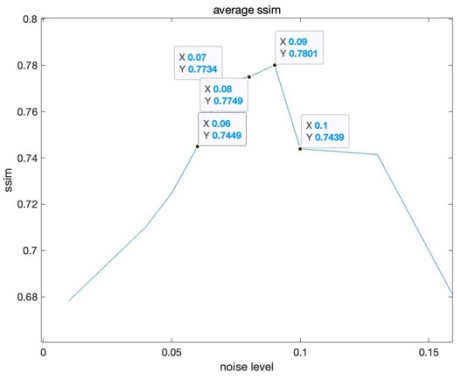


Figure 32: Reconstruction average SSIM vs denoising level

We can see that all the networks with denoising level $\boldsymbol{\sigma_2}$ within the range of 0.07 ~ 0.09 can achieve better reconstruction results in the hybrid algorithm. We will show the best results when the denoising level is 0.08：

figure	1	2	3	4	5	6	7	8	9	10	11	12
rsnr	9.6399	12.493	7.5772	4.4581	11.682	12.182	12.412	18.457	14.759	7.8968	9.027	11.1
ssim	0.47562	0.80183	0.68391	0.34464	0.77098	0.75858	0.79693	0.92987	0.90996	0.79369	0.8484	0.88949
time(sec)	125.54	219.75	75.854	170.64	127.86	242.52	240.97	156.49	188.97	119.3	240.63	96.11
figure	13	14	15	16	17	18	19	20	21	22	23	24
rsnr	10.303	11.478	18.645	7.006	10.116	10.869	10.049	9.6409	17.393	7.2595	16.763	9.0004
ssim	0.83104	0.80775	0.91754	0.67666	0.78711	0.86165	0.772	0.79418	0.80333	0.78796	0.89305	0.6607
time(sec)	62.594	243.48	155.43	44.428	64.496	229.78	239.4	123.82	93.02	168.39	107.95	117.17
average rsnr:	11.259		denoising level	0.08								
average ssim:	0.77487		delta:	1.00E-07								
average time:	152.2746667											

Table 3: Reconstruction result for constrained objective function at denoising level = 0.08 with patch size = 512

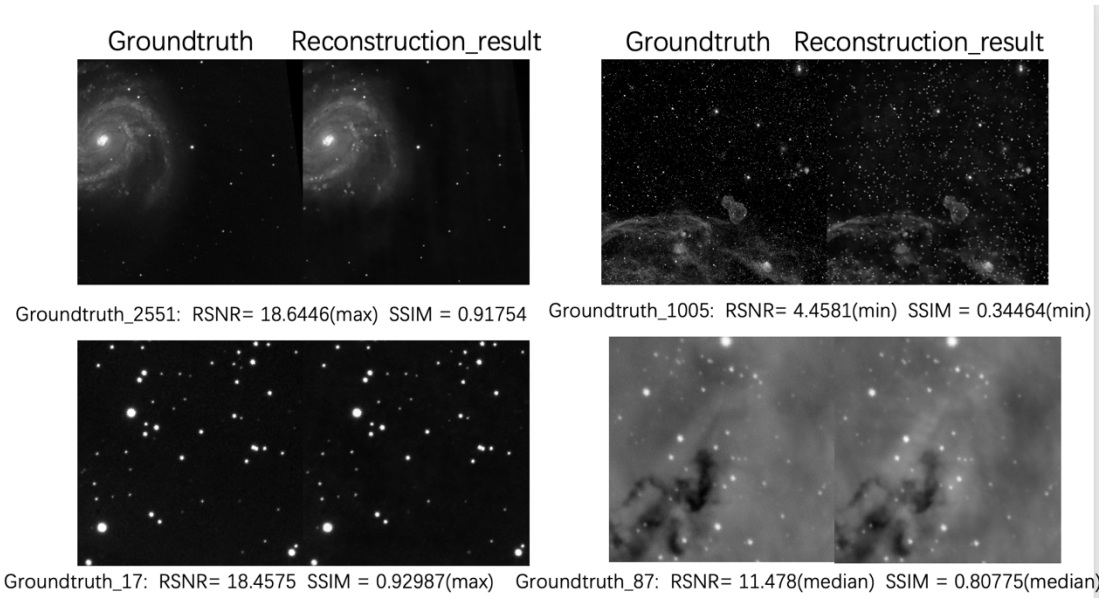


Figure 34: Reconstruction images for constrained objective function at denoising level = 0.08 with patch size = 512

We can see that our PnP algorithm achieves outstanding reconstruction results, not only for images containing only stars but also for nebulae and galaxies with complex texture structures.

However, as for the poor reconstruction result of the 1005th graph, we believe that it is not caused by the algorithm. When we look carefully at the original image, we find that there are randomly distributed noise points on the surface of this image, which are not found in other images. In other words, this picture is not the ground truth in the true sense. It is because of the emergence of these noise points that the learned network apriori thinks that they are stars. We can see that the noise in the reconstructed image is obviously brighter. It is precise because of this that there is a big gap between the reconstruction and the original image.

For PnP algorithm of unconstrained objective function:

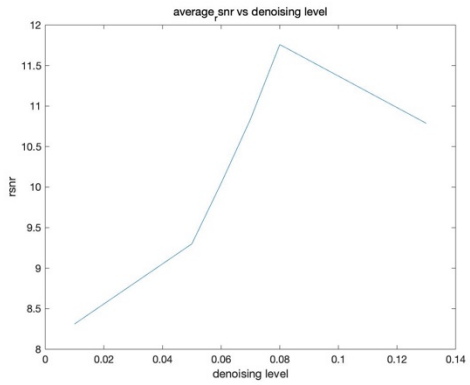


Figure 35: Reconstruction average RSNR vs denoising level

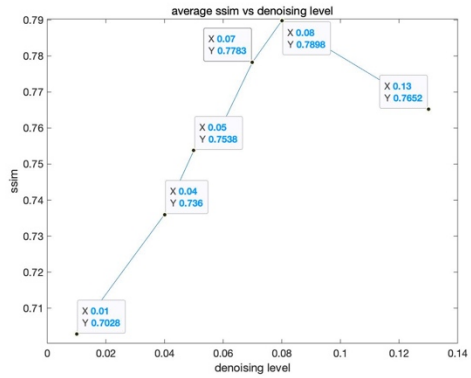
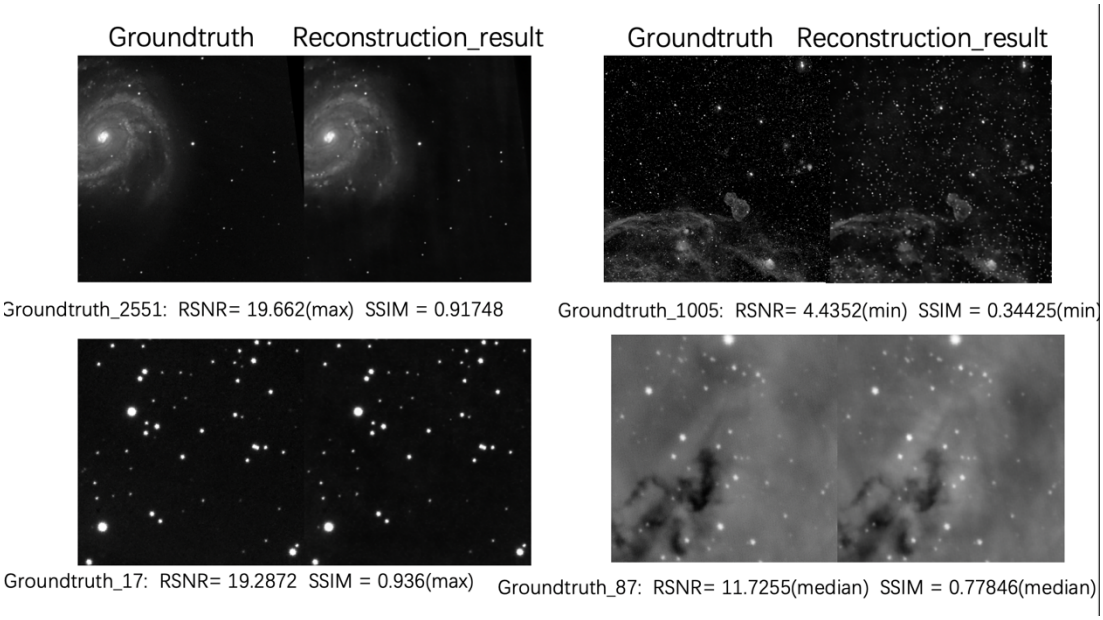


Figure 36: Figure 32: Reconstruction average SSIM vs denoising level

We can see that for the all the unconstrained objective function, networks with denoising level $\boldsymbol{\sigma_2}$ within the range of 0.07 ~ 0.09 can achieve better reconstruction results in the hybrid algorithm which is similar to what we found before in the constrained objective function. We will show the best results when the denoising level is 0.08：

image	1	2	3	4	5	6	7	8	9	10	11	12
rsnr	9.6887	12.52	10.763	4.4352	12.768	12.175	12.416	19.287	14.853	8.4869	9.0465	11.11
ssim	0.47843	0.80202	0.81425	0.34425	0.79931	0.75848	0.7968	0.93609	0.91094	0.81638	0.84797	0.88965
time(s)	297.38	296.87	296.87	84.074	297.63	296.71	296.08	297.27	295.34	295.53	295.62	295.45
image	13	14	15	16	17	18	19	20	21	22	23	24
rsnr	10.302	11.477	19.662	9.0724	10.116	10.795	10.049	9.6412	17.396	7.2584	17.169	11.726
ssim	0.83102	0.80771	0.91748	0.72203	0.78713	0.85969	0.77202	0.7942	0.80339	0.7879	0.89911	0.77846
time(s)	295.6	295.18	295.25	296.61	295.65	295.42	295.68	298.75	302.18	304.11	321.79	357.61
average_rsnr	11.759											
average_time	0.78978											



5.4 Comparison

reconstruction result				
method		average_rsnr	average_ssim	verage_reconstruction_time(s
ADMM	Constrained obj function	5.8018	0.53813	23.97415833
	Unconstrained obj function	6.6334	0.62814	94.88175
PnP	Constrained obj function	11.259	0.77487	152.2746667
	Unconstrained obj function	11.759	0.78978	291.6105833
pure nearual network method		10.2582712	0.712227357	1.47204231

From this, we can see that the results of the PNP hybrid algorithm based on learning prior are much better than those of the ADMM algorithm based on sparse prior. The results of the PNP algorithm with careful parameter adjustment are even better than those of pure neural network algorithm based on the mapping. However, the problem is that for PnP hybrid algorithm, it takes a long time to reconstruct without considering the training time.

Here, we use blue to highlight the results of PNP algorithm dominance and green to highlight the results of pure neural network algorithm dominance, as shown in the following figure:

figure	1	2	3	4	5	6	7	8	9	10	11	12
rsnr	9.6399	12.493	7.5772	4.4581	11.682	12.182	12.412	18.457	14.759	7.8968	9.027	11.1
ssim	0.47562	0.80183	0.68391	0.34464	0.77098	0.75858	0.79693	0.92987	0.90996	0.79369	0.8484	0.88949
time(sec)	125.54	219.75	75.854	170.64	127.86	242.52	240.97	156.49	188.97	119.3	240.63	96.11
figure	13	14	15	16	17	18	19	20	21	22	23	24
rsnr	10.303	11.478	18.645	7.006	10.116	10.869	10.049	9.6409	17.393	7.2595	16.763	9.0004
ssim	0.83104	0.80775	0.91754	0.67666	0.78711	0.86165	0.772	0.79418	0.80333	0.78796	0.89305	0.6607
time(sec)	62.594	243.48	155.43	44.428	64.496	229.78	239.4	123.82	93.02	168.39	107.95	117.17
average rsnr:	11.259		denoising l	0.08								
average ssim:	0.77487		delta:	1.00E-07								
average time:	152.2746667											

figure	1	2	3	4	5	6	7	8	9	10	11	12
rsnr	7.93729751	10.95776186	14.57629	4.0905469	11.2348	8.17068	12.58	11.6788	5.46043	9.05097	10.1937	10.6387
ssim	0.409684037	0.772861297	0.734821	0.2959822	0.74818	0.68842	0.70588	0.8051	0.75565	0.74234	0.73133	0.81136
time(sec)	2.498546197	1.458230537	1.333321	1.3692391	1.3601	1.31504	1.34948	1.35298	1.35534	1.39318	1.39795	1.39234
figure	13	14	15	16	17	18	19	20	21	22	23	24
rsnr	6.998711228	15.55451515	9.755264	7.3164037	12.8176	12.9181	12.0715	10.3664	9.58764	10.3659	8.71479	13.1617
ssim	0.730183392	0.804971383	0.816307	0.5712616	0.77349	0.78131	0.76123	0.76393	0.6637	0.80373	0.74257	0.67917
time(sec)	1.488636769	1.44277009	1.43857	1.4263788	1.44412	1.54334	1.44798	1.47438	1.45568	1.45012	1.59024	1.55106
average rsnr:	10.25827119											
average ssim:	0.712227357											
average time:	1.47204231											

Table 4: Reconstruction result for PnP algorithm and pure neural network

6 Conclusions

In this project, we derive and implement ADMM algorithms with different objective functions. Besides, we successfully implement PnP hybrid algorithm based on ADMM algorithm. The results confirm that our PnP hybrid algorithm gets much better reconstruction results than ADMM algorithm. With careful parameter adjustment, the results of the PNP algorithm are even better than those of pure neural network algorithm.

During the implementation of the ADMM algorithm, we explore the range of ρ, which can influence the reconstruction result. In the process of implementing PnP algorithm, we find that the larger network input size will bring better denoising results and reconstruction results. In addition, we also explore the influence of different denoising level $\boldsymbol{\sigma_2}$ to the reconstruction result.

Although the PNP algorithm can bring a guaranteed reconstruction effect, it has two disadvantages. On the one hand, because the denoising operator is nonconvex and non-differentiable, it means that the convergence of PNP algorithm is difficult to analyze. On the other hand, PNP algorithm relies heavily on ADMM algorithm.

Recently, Yaniv Romano et al. proposed a more robust and more flexible hybrid algorithm which is called regularized by denoising (RED)[14]. RED can be flexibly combined with a variety of optimization algorithms and has been shown to converge to the global optimum. This provides us with the direction for the next phase of the project.

7 Bibliography

- *B31XO_2019-2020_Sampling and Computational imaging lectures from Prof. Yves Wiaux*

8 References

1. Thompson, A., J. Moran, and J.G. Swenson, *Interferometry and Synthesis in Radio Astronomy*. Vol. -1. 1991.
2. Donoho, D.L., *Compressed sensing.* IEEE Transactions on Information Theory, 2006. 52(4): p. 1289-1306.
3. Jin, K.H., et al., *Deep Convolutional Neural Network for Inverse Problems in Imaging.* IEEE Transactions on Image Processing, 2017. 26(9): p. 4509-4522.
4. Venkatakrishnan, S.V., C.A. Bouman, and B. Wohlberg. *Plug-and-Play priors for model based reconstruction*. 2014.
5. Ronneberger, O., P. Fischer, and T. Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation.* Computer Science, 2015.
6. Candes, E.J. and T. Tao, *Decoding by linear programming.* IEEE Transactions on Information Theory, 2005. 51(12): p. 4203-4215.
7. Candès, E. and J. Romberg, *Sparsity and incoherence in compressive sampling.* Inverse Problems, 2007. 23(3): p. 969-985.
8. Bruckstein, A.M., D.L. Donoho, and M. Elad, *From Sparse Solutions of Systems of Equations to Sparse Modeling of Signals and Images.* Siam Review, 2009. 51(1): p. 34-81.
9. MathWorks. Available from: https://ww2.mathworks.cn/help/wavelet/ug/matching-pursuit-algorithms.html.
10. Cai, T.T. and L. Wang, *Orthogonal Matching Pursuit for Sparse Signal Recovery With Noise.* IEEE Transactions on Information Theory, 2011. 57(7): p. 4680-4688.
11. WikiPedia. *Convex Funcion*. Available from: https://en.wikipedia.org/wiki/Convex_function.
12. Snipes, M., *Linear and Nonlinear Programming.* American Mathematical Monthly, 2013. 120.
13. Stephen, B., et al., *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. 2011: now. 1.
14. Romano, Y., M. Elad, and P. Milanfar, *The Little Engine that Could: Regularization by Denoising (RED).* SIAM Journal on Imaging Sciences, 2016. 10.
15. WikiPedia. *proximal operator*. Available from: https://en.wikipedia.org/wiki/Proximal_operator.

16. Parikh, N. and S. Boyd, *Proximal Algorithms.* Foundations and Trends® in Optimization, 2014. 1(3): p. 127-239.
17. Combettes, P. and V. Wajs, *Signal Recovery by Proximal Forward-Backward Splitting.* Multiscale Modeling & Simulation - MULTISCALE MODEL SIMUL, 2005. 4.
18. Chan, S.H., X. Wang, and O.A. Elgendy, *Plug-and-Play ADMM for Image Restoration: Fixed-Point Convergence and Applications.* IEEE Transactions on Computational Imaging, 2017. 3(1): p. 84-98.
19. Gavaskar, R. and K. Chaudhury, *On the Proof of Fixed-Point Convergence for Plug-and-Play ADMM*. 2019.
20. Zhang, K., et al., *Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising.* IEEE Transactions on Image Processing, 2017. 26(7): p. 3142-3155.
21. Burger, H.C., C.J. Schuler, and S. Harmeling. *Image denoising: Can plain neural networks compete with BM3D?* in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012.
22. He, K., et al. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. in *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015.
23. Dellinger, J. *Weight Initialization in Neural Networks: A Journey From the Basics to Kaiming*. 2019; Available from: https://towardsdatascience.com/weight-initialization-in-neural-networks-a-journey-from-the-basics-to-kaiming-954fb9b47c79.

Appendices

1. Derivation and proof

- prove $\boldsymbol{u^* = prox_{l_{C(0,\varepsilon)}}(z) = min\{\varepsilon, \|z\|_2\} * \frac{z}{\|z\|_2}}$

where indicator function $\boldsymbol{r(u) = l_{C(0,\varepsilon)}(u) = \begin{cases} 0 & \|u\|_2 \le \varepsilon \\ +\infty & otherwise \end{cases}}$

Indicator function is not a differentiable term owing to the discontinuity. In the geometric sense, the indicator function $\boldsymbol{l_{C(0,\varepsilon)}(u)}$ represents a circle set $\boldsymbol{C}$ with the centre at 0 and the radius of noise boundary $\boldsymbol{\varepsilon}$.

we have the following expression and transformation for the proximal operator of $\boldsymbol{r(u)}$:

$$\boldsymbol{u^* = prox_{l_{C(0,\varepsilon)}}(z) = \arg\min\left\{l_{C(0,\varepsilon)}(u) + \frac{1}{2}\|u - z\|_2^2\right\} \quad u \in R^n}$$

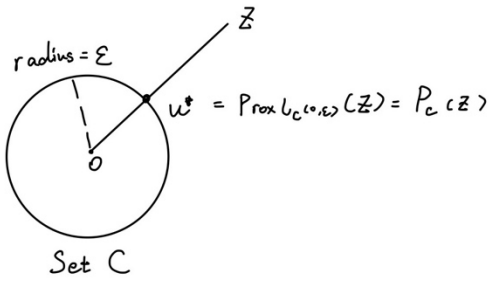
$$\boldsymbol{u^* = prox_{l_{C(0,\varepsilon)}}(z) = argmin\left\{\frac{1}{2}\|u - z\|_2^2\right\} \quad u \in C^n}$$

$$\boldsymbol{because \quad P_C(z) = argmin\{\|u - z\|_2^2\} \quad u \in C^n}$$

$$\boldsymbol{therefore \quad u^* = prox_r(z) = P_C(z) \quad u \in C^n}$$

Where $\boldsymbol{P_C(z)}$ represents the project operator, the mathematical meaning of these transformations is:

The proximal operator at point z of the indicator function $\boldsymbol{l_{C(0,\varepsilon)}(u)}$ is equal to the projection of point z onto the set $\boldsymbol{C}$, which the indicator function represents. This means that we can obtain the analytic solution of the proximal operator by the solution of the projection operator. From the figure below, we can get the specific projection value：



If the point z is outside the circuit $\|\boldsymbol{z}\|_2 > \boldsymbol{\varepsilon}$, we can get $\boldsymbol{u}^* = \boldsymbol{\varepsilon} * \frac{\boldsymbol{z}}{\|\boldsymbol{z}\|_2}$

If the point z is inside the circuit $\|\boldsymbol{z}\|_2 < \boldsymbol{\varepsilon}$, we can get $\boldsymbol{u}^* = \boldsymbol{z}$.

In summary, $\boldsymbol{u}^* = \boldsymbol{prox}_{\boldsymbol{l}_{C(0,\varepsilon)}}(\boldsymbol{z}) = \boldsymbol{min}\{\boldsymbol{\varepsilon}, \|\boldsymbol{z}\|_2\} * \frac{\boldsymbol{z}}{\|\boldsymbol{z}\|_2}$

- Prove $\boldsymbol{u}^* = \boldsymbol{prox}_{\boldsymbol{\lambda}\|\boldsymbol{u}\|_2^2}(\boldsymbol{z}) = \frac{\boldsymbol{z}}{\boldsymbol{2\lambda+1}}$

we have the following expression for the proximal operator of $\boldsymbol{\ell_2}$ norm squared $\boldsymbol{r(u)}$:

$$\boldsymbol{u}^* = \boldsymbol{prox}_{\boldsymbol{\lambda}\|\boldsymbol{u}\|_2^2}(\boldsymbol{z}) = \boldsymbol{argmin}\left\{\boldsymbol{\lambda}\|\boldsymbol{u}\|_2^2 + \frac{\boldsymbol{1}}{\boldsymbol{2}}\|\boldsymbol{u}-\boldsymbol{z}\|_2^2\right\} \quad \boldsymbol{u} \in \boldsymbol{R^n}$$

We notice that $\boldsymbol{r(u)} = \boldsymbol{\lambda}\|\boldsymbol{u}\|_2^2$ is a differentiable term and $\frac{\boldsymbol{1}}{\boldsymbol{2}}\|\boldsymbol{u}-\boldsymbol{z}\|_2^2$ is also a differentiable term which means the objective function $\boldsymbol{f}$ is differentiable.

Therefore, we can get the minimum point $\boldsymbol{u}^*$ by calculating the zero gradient point directly:

$$\boldsymbol{u}^* = \boldsymbol{prox}_{\boldsymbol{\lambda}\|\boldsymbol{u}\|_2^2}(\boldsymbol{z}) = \mathbf{arg}\{\boldsymbol{\nabla f}(\boldsymbol{u}^*) = \boldsymbol{0}\} \quad \boldsymbol{u} \in \boldsymbol{R^n}$$

$$\boldsymbol{2\lambda u}^* + \boldsymbol{u}^* - \boldsymbol{z} = \boldsymbol{0}$$

$$\boldsymbol{u}^* = \frac{\boldsymbol{z}}{\boldsymbol{2\lambda+1}}$$

In summary, $\boldsymbol{u}^* = \boldsymbol{prox}_{\boldsymbol{\lambda}\|\boldsymbol{u}\|_2^2}(\boldsymbol{z}) = \frac{\boldsymbol{z}}{\boldsymbol{2\lambda+1}}$

- Prove $\boldsymbol{u}^* = \boldsymbol{prox}_{\boldsymbol{\lambda}\|\boldsymbol{u}\|_1}(\boldsymbol{z}) = \mathbf{max}\{|\boldsymbol{z}| - \boldsymbol{\lambda}, \boldsymbol{0}\} \cdot \boldsymbol{sign}(\boldsymbol{z}) = \boldsymbol{shrink}(\boldsymbol{z}, \boldsymbol{\lambda})$

we have the following expression for the proximal operator of $\boldsymbol{\ell_1}$ norm $\boldsymbol{r(u)}$:

$$\boldsymbol{u^* = prox_{\lambda\|u\|_1}(z) = argmin\left\{\lambda\|u\|_1 + \frac{1}{2}\|u - z\|_2^2\right\}\ u \in R^n}$$

Remember that one of the important properties of the convex optimization problem which we discussed in last chapter is that For non-differential and convex object function, the necessary and sufficient condition for $\boldsymbol{u^*}$ to be the minimum is that the sub-gradient at $\boldsymbol{u^*}$ contains zero-point $\boldsymbol{0 \in \partial f(u^*)}$.

Here, we calculate the sub-gradient of our objective function, we can get:

$$\boldsymbol{\partial f(u) = \begin{cases} \lambda + u - z & if\ u > 0 \\ \lambda * [-1, 1] - z & if\ u = 0 \\ -\lambda + u - z & if\ u < 0 \end{cases}}$$

By imposing on the optimality condition $\boldsymbol{0 \in \partial f(u^*)}$, we can get the following relationships:

$$\boldsymbol{0 = \begin{cases} \lambda + u - z & if\ u > 0 \\ \lambda * [-1, 1] - z & if\ u = 0 \\ -\lambda + u - z & if\ u < 0 \end{cases}}$$

Then, we can get the expression for $\boldsymbol{u^*}$ at the different conditions

$$\boldsymbol{u^* = \begin{cases} z - \lambda & if\ z > \lambda \\ 0 & if - \lambda < z < \lambda \\ z + \lambda & if\ z < -\lambda \end{cases}}$$

In summary, $\boldsymbol{u^* = prox_{\lambda\|u\|_1}(z) = \max\{|z| - \lambda, 0\} \cdot sign(z) = shrink(z, \lambda)}$

- Prove $\boldsymbol{prox_{\rho^{-1}\delta * r(\psi^\dagger x)}(z) = \psi\, prox_{\rho^{-1}\delta * r(x)}(\psi^\dagger z)}$

$$\boldsymbol{prox_{\rho^{-1}\delta * r(\psi^\dagger x)}(z) = argmin\ \rho^{-1}\delta * r(\psi^\dagger x) + \frac{1}{2}\|x - z\|_2^2}$$

$$\boldsymbol{prox_{\rho^{-1}\delta * r(\psi^\dagger x)}(z) = argmin\ \rho^{-1}\delta * r(\psi^\dagger \psi w) + \frac{1}{2}\|\psi w - z\|_2^2\ \ with\ \psi w = x}$$

$$\boldsymbol{prox_{\rho^{-1}\delta * r(\psi^\dagger x)}(z) = argmin \rho^{-1}\delta * \psi * r(w) + \frac{1}{2}\left\|\psi(w - \psi^\dagger z)\right\|_2^2}$$

$$\boldsymbol{prox_{\rho^{-1}\delta * r(\psi^\dagger x)}(z) = \psi *\ argmin\ \rho^{-1}\delta * \psi * r(w) + \frac{1}{2}\left\|(w - \psi^\dagger z)\right\|_2^2}$$

$$\boldsymbol{prox_{\rho^{-1}\delta * r(\psi^\dagger x)}(z) = \psi\, prox_{\rho^{-1}\delta * r(x)}(\psi^\dagger z)}$$

2. MATLAB Code

- Code for Bayesian optimizer

Here we will describe the part of the code where we apply the Bayesian optimizer to network training. The main program of the Bayesian optimization algorithm is the Bayes.m.

```
DataPath = './good';
patchSize = 512;
channels = 32;
networkDepth = 17;
ratio = 0.01 ;
imds = imageDatastore(DataPath,'ReadFcn',@NormalizeImageResize);
fileNums = numel(imds.Files);
nFiles = length(imds.Files);
RandIndices = randperm(nFiles);
ratio2num = round(ratio*nFiles);
validation_indices = RandIndices(1:ratio2num);
validation_set = subset(imds,validation_indices);

train_indices= RandIndices(ratio2num+1:nFiles);
train_set = subset(imds,train_indices);

optimVars = [
    optimizableVariable('MaxEpochs',[6,20],'Type','integer')
    optimizableVariable('LearnRateDropPeriod',[2,10],'Type','integer');
    optimizableVariable('InitialLearnRate',[5e-2,5e-3],'Transform','log')];

ObjFcn = makeObjFcn(train_set,validation_set,patchSize,channels,networkDepth);

BayesObject = bayesopt(ObjFcn,optimVars, ...
    'MaxTime',Inf, ...
    'MaxObjectiveEvaluations',30,...
    'IsObjectiveDeterministic',false, ...
    'AcquisitionFunctionName','expected-improvement-plus',...
    'NumSeedPoints',4,...
    'ExplorationRatio',0.5,...
     'Verbose',1,...
     'UseParallel',false);
```

In the Bayes.m, we first load the data files into the image datastore structure. Then, we will divide the data files into training set and validation set according to the ratio. Followed by that, we will choose the variables which need to optimize and determine the options of Bayesian optimizer. Finally, we deliver the parameters and options to the optimizer, which will tirelessly minimize the value of the objective function.

makeObjFcn.m

```
function ObjFcn = makeObjFcn(train_set,validation_set,patchSize,channels,networkDepth)

ObjFcn = @valErrorFun;

function [reconstruction_result,con,fileName] = valErrorFun(optVars)

    train_source = denoisingImageDatastore(train_set, ...
    'patchSize', patchSize, ...
    'ChannelFormat', channelFormat, ...
    'GaussianNoiseLevel', noiseStd, ...
    'PatchesPerImage', patchesPerImage);

    validation_source = denoisingImageDatastore(validation_set, ...
    'patchSize', patchSize, ...
    'ChannelFormat', channelFormat, ...
    'GaussianNoiseLevel', noiseStd, ...
    'PatchesPerImage', patchesPerImage);

layers = structure(patchSize, channels, networkDepth);
```

```
options = trainingOptions('adam',...
    'MaxEpochs',optVars.MaxEpochs,...
    'MiniBatchSize',32,...
    'ValidationData',validation_source,...
    'ValidationPatience',Inf,...
    'Verbose',false,...
    'Shuffle','every-epoch',...
    'InitialLearnRate',optVars.InitialLearnRate, ...
    'LearnRateSchedule','piecewise',...
    'LearnRateDropFactor',0.1,...
    'LearnRateDropPeriod',optVars.LearnRateDropPeriod,...
    'GradientThreshold',1,...
    'ExecutionEnvironment','multi-gpu');

     trainedNet = trainNetwork(train_source,layers,options);
     reconstruction_result = -1*(cal_denoise_test(trainedNet));
     fileName =['./project/net/bayes/',num2str(reconstruction_result),'.mat'];
     save(fileName,'trainedNet','reconstruction_result','options')
     con = [];
end
end
```

The makeObjFcn.m is the objective function code which consists of two parts: one is to train the network using the optimization variable, and the other is to pass the trained network into the cal_denoise_test.m function which can analyze the denoising performance of the network and return the value of the denoising result. We will pass the denoising result to the optimizer by multiplying it by minus one as the return value of the objective function. The reason why we need to multiply by minus one is that we want the higher value of the denoising result, but the goal of the Bayesian optimizer is to minimize the value of the target function.

Structure.m

```
function layers = structure(patchSize, channels, networkDepth)
    layers = [];
    b_min = 0.025;

    % Input
    input = imageInputLayer([patchSize patchSize channels], ...
        'Normalization', 'none', ...
        'Name', 'Input');

    layers = [layers, input];

    % Conv+ReLU
    conv = convolution2dLayer(3, 64, ...
        'Stride', 1, ...
        'Padding', 'same', ...
        'WeightLearnRateFactor', 1, ...
        'WeightL2Factor', 1, ...
        'BiasLearnRateFactor', 1, ...
        'BiasL2Factor', 0, ...
        'Name', 'Conv1');
    conv.Weights = sqrt(2/(9*64))*randn(3,3,channels,64,'single');
    conv.Bias = zeros(1,1,64,'single');

    relu = reluLayer(...
        'Name', 'ReLU1');

    layers = [layers, conv, relu];
```

```
    % Conv+BN+ReLU
    for i = 2:networkDepth-1
        conv = convolution2dLayer(3, 64, ...
            'Stride', 1, ...
            'Padding', 'same', ...
            'WeightLearnRateFactor', 1, ...
            'WeightL2Factor', 1, ...
            'BiasLearnRateFactor', 0, ...
            'BiasL2Factor', 0, ...
            'Name', ['Conv', num2str(i)]);
        conv.Weights = sqrt(2/(9*64))*randn(3,3,64,64,'single');
        conv.Bias = zeros(1,1,64,'single');

        bnorm = batchNormalizationLayer(...
            'Scale', clipping(sqrt(2/(9*64))*randn(1,1,64,'single'), b_min), ...
            'ScaleLearnRateFactor', 1, ...
            'ScaleL2Factor', 0, ...
            'Offset', zeros(1,1,64,'single'), ...
            'OffsetLearnRateFactor', 1, ...
            'OffsetL2Factor', 0, ...
            'Name', ['BNorm', num2str(i)]);

        relu = reluLayer(...
            'Name', ['ReLU', num2str(i)]);

        layers = [layers, conv, bnorm, relu];
    end
```

```
    % Conv
    conv = convolution2dLayer(3, channels, ...
        'Stride', 1, ...
        'Padding', 'same', ...
        'WeightLearnRateFactor', 1, ...
        'WeightL2Factor', 1, ...
        'BiasLearnRateFactor', 1, ...
        'BiasL2Factor', 0, ...
        'Name', ['Conv', num2str(networkDepth)]);
    conv.Weights = sqrt(2/(9*64))*randn(3,3,64,channels,'single');
    conv.Bias = zeros(1,1,channels,'single');

    layers = [layers, conv];

    % Regression
    regression = regressionLayer(...
        'Name', 'Output');

    layers = [layers, regression];
end

function A = clipping(A, b)
    A(A>=0 & A<b) = b;
    A(A<0 & A>-b) = -b;
end
```

The code of structure.m mainly describes the structure of the network. We need to call this function every time we train the network.

3. Reconstruction result.

The reconstruction results for analyzing patch-size.

- Reconstruction for the denoising network with patch-size = 32 and denoising level = 0.07

image	1	2	3	4	5	6	7	8	9	10	11	12
rsnr	8.8938	6.4767	6.8003	5.9691	10.124	10.214	7.9367	16.15	4.5067	5.3846	6.3105	10.94
ssim	0.47074	0.72269	0.65574	0.45045	0.70485	0.68584	0.62541	0.89454	0.68573	0.6261	0.56346	0.88162
time	343.82	379.06	380.31	377.33	90.097	221.88	99.063	376.1	328.26	381.33	377.13	78.326
image	13	14	15	16	17	18	19	20	21	22	23	24
rsnr	9.6476	8.8293	6.2297	4.3835	6.837	7.4631	8.1249	7.831	14.506	5.9359	16.229	13.532
ssim	0.77248	0.6764	0.79106	0.53504	0.51628	0.76207	0.59076	0.59238	0.70727	0.66858	0.8114	0.70324
time	377.02	375.78	105.5	386.66	381.38	122.38	377.76	377.52	33.012	93.482	267.63	201.58
average_rsnr	8.719											
average_ssim	0.67059											

- Reconstruction for the denoising network with patch-size = 64 and denoising level = 0.07

image	1	2	3	4	5	6	7	8	9	10	11	12
rsnr	9.2342	8.5192	7.3589	4.7454	11.487	11.536	14.814	16.094	5.3221	7.7584	8.0855	2.39
ssim	0.43813	0.7703	0.6595	0.35886	0.76749	0.74395	0.7954	0.89555	0.79595	0.79595	0.79502	0.67692
time	622.96	281.25	153.67	786.68	373.18	680.68	680.84	144.05	559.12	379.86	632.45	138.35
image	13	14	15	16	17	18	19	20	21	22	23	24
rsnr	9.9687	11.293	8.1186	6.1565	9.6799	8.3931	9.7542	9.0777	15.567	6.8124	20.792	10.082
ssim	0.82366	0.79539	0.81372	0.65313	0.74176	0.76813	0.73885	0.73931	0.75467	0.76422	0.87791	0.67203
time	328.43	772.78	553.48	351.35	706.05	326.58	156.43	275.93	773.64	513.42	309.67	184.32
average_rsnr	9.71											
average_ssim	0.73483											

- Reconstruction for the denoising network with patch-size = 128 and denoising level = 0.07

image	1	2	3	4	5	6	7	8	9	10	11	12
rsnr	9.0265	9.2437	11.28	9.2534	4.4312	12.317	6.077	13.743	16.787	6.1411	8.3651	8.316
ssim	0.82901	0.46833	0.73254	0.79925	0.34919	0.78494	0.59673	0.80315	0.91691	0.81456	0.80078	0.809
time	265.13	258.94	259.18	258.39	259.38	254.4	254.41	254.27	254.74	259.76	259.47	259.7
image	13	14	15	16	17	18	19	20	21	22	23	24
rsnr	13.036	8.4556	5.8649	10.406	7.634	6.5267	11.176	8.1938	6.8952	16.19	19.429	13.607
ssim	0.88381	0.78879	0.64988	0.84233	0.67801	0.731	0.81615	0.76784	0.74163	0.77496	0.89732	0.74733
time	260.08	259.95	256.28	254.72	254.84	255.11	255.23	255.03	255.02	255.17	254.4	256.46
average_rsnr	10.1											
average_ssim	0.75098											

- Reconstruction for the denoising network with patch-size = 256 and denoising level = 0.07

image	1	2	3	4	5	6	7	8	9	10	11	12
rsnr	9.1632	8.0572	7.4335	5.2011	11.278	7.9805	12.787	15.589	9.6361	7.7391	8.8031	16.607
ssim	0.47827	0.76761	0.68975	0.38363	0.75204	0.74085	0.77424	0.8927	0.87882	0.79076	0.7704	0.95993
time	163.32	99.931	46.775	223.97	86.416	159.76	224.06	84.918	224.1	148.29	23.006	53.655
image	13	14	15	16	17	18	19	20	21	22	23	24
rsnr	9.8167	11.467	13.799	7.5485	10.979	9.2991	10.282	10.08	18.199	6.52	14.884	11.109
ssim	0.83986	0.80946	0.8727	0.70465	0.80867	0.84821	0.79707	0.80964	0.82023	0.78427	0.88501	0.78357
time	133.52	195.28	223.37	223.65	61.01	174.43	81.332	195.64	128.91	99.482	224.34	156.95
average_rsnr	10.594											
average_ssim	0.77676											

- Reconstruction for the denoising network with patch-size = 512 and denoising level = 0.07
-

image	1	2	3	4	5	6	7	8	9	10	11	12
rsnr	10.079	7.0986	8.4008	4.8693	11.598	11.653	12.696	17.793	10.303	8.056	10.645	10.789
ssim	0.49587	0.74477	0.78404	0.35713	0.76508	0.75142	0.78818	0.92523	0.88109	0.79827	0.85668	0.8823
time	207.81	46.297	191.75	200.08	114.41	237.79	237.88	173.08	87.59	238.57	240.38	104.12
image	13	14	15	16	17	18	19	20	21	22	23	24
rsnr	10.515	10.209	16.635	6.7797	8.7798	9.0974	9.0056	8.3334	17.546	6.8869	16.914	11.591
ssim	0.83469	0.79477	0.90872	0.68122	0.74266	0.83125	0.74471	0.74795	0.81063	0.7817	0.884	0.76942
time	185.17	227.98	181.82	92.762	88.299	71.098	240.47	142.2	129.44	42.079	43.967	238.77
average_rsnr	10.678											
average_ssim	0.77341											

The reconstruction results for analyzing denoising level:

- Reconstruction for the denoising network with patch-size = 512 and denoising level = 0.01

image	1	2	3	4	5	6	7	8	9	10	11	12
rsnr	8.9601	5.2003	6.5944	6.3107	10.083	6.6653	6.0575	11.892	3.8416	7.1595	6.4592	15.306
ssim	0.48643	0.65814	0.68011	0.50141	0.65578	0.64427	0.5389	0.77083	0.68346	0.73973	0.706	0.91014
time(s)	136.08	20.296	118.08	25.997	83.554	93.886	39.9	32.042	76.392	83.252	116.06	131.35
image	13	14	15	16	17	18	19	20	21	22	23	24
rsnr	7.2186	7.4688	6.9611	6.583	5.2641	7.7293	6.7809	5.6052	16.493	4.9957	9.008	7.4356
ssim	0.70244	0.70093	0.76607	0.61375	0.64194	0.75634	0.71404	0.67737	0.75924	0.66763	0.7116	0.59329
time(s)	32.946	53.001	76.547	83.594	44.53	154.49	54.792	67.91	46.782	101.76	221.69	28.714
average_rsnr	7.7531											
average_ssim	0.67833											

- Reconstruction for the denoising network with patch-size = 512 and denoising level = 0.04

image	1	2	3	4	5	6	7	8	9	10	11	12
rsnr	9.7203	6.0578	8.3484	6.3266	10.172	6.5056	8.2075	13.385	8.4228	5.9681	7.0864	14.984
ssim	0.50071	0.6626	0.77027	0.45943	0.69384	0.66069	0.66919	0.83822	0.85555	0.66541	0.75447	0.89367
time(s)	275.31	68.988	179.67	203.11	133.65	125.12	137.97	271.23	216.21	54.267	58.229	225.32
image	13	14	15	16	17	18	19	20	21	22	23	24
rsnr	6.5283	10.135	12.312	6.8363	5.6362	9.2319	5.8749	5.2302	18.249	4.8061	10.168	11.248
ssim	0.65669	0.78687	0.85834	0.66362	0.7095	0.8326	0.68362	0.66566	0.8284	0.67203	0.82453	0.74999
time(s)	52.304	129.28	271.11	147.4	119.41	201.71	90.535	70.102	88.073	270.31	64.784	74.916
average_rsnr	8.81001667											
average_ssim	0.7231625											

- Reconstruction for the denoising network with patch-size = 512 and denoising level = 0.05

image	1	2	3	4	5	6	7	8	9	10	11	12
rsnr	9.2963	5.8654	6.8032	5.088	11.679	6.8568	6.3787	16.565	10.457	7.9444	8.2747	12.998
ssim	0.48215	0.69454	0.72728	0.37953	0.76726	0.67905	0.60727	0.91295	0.89219	0.80069	0.83131	0.91739
time(s)	128.77	95.43	205.43	52.556	197.89	208.3	117.76	198.82	173.44	180.55	144.96	49.32
image	13	14	15	16	17	18	19	20	21	22	23	24
rsnr	8.0426	6.0948	5.1695	6.1069	5.3598	9.2292	5.2856	6.0479	16.037	4.371	8.7107	9.5424
ssim	0.75819	0.6883	0.7736	0.65665	0.6892	0.85231	0.57606	0.72365	0.77087	0.67727	0.79719	0.74128
time(s)	60.575	230.85	17.59	77.964	56.841	300.24	31.406	192.74	28.248	201.4	303.42	303.5
average_rsnr	8.25849583											
average_ssim	0.72484083											

- Reconstruction for the denoising network with patch-size = 512 and denoising level = 0.06

image	1	2	3	4	5	6	7	8	9	10	11	12
rsnr	9.7209	7.1832	6.555	5.3772	10.034	11.934	12.179	16.081	5.5259	7.0937	7.9798	6.7198
ssim	0.49336	0.74888	0.66452	0.38915	0.74748	0.75901	0.78631	0.90282	0.75873	0.74684	0.82717	0.79629
time(s)	267.57	44.331	31.665	265.87	38.965	165.62	97.131	58.024	131.57	68.574	263.24	80.618
image	13	14	15	16	17	18	19	20	21	22	23	24
rsnr	10.448	9.8438	9.3185	6.2584	8.3204	7.6399	8.7788	8.0127	17.838	6.8098	17.803	9.9712
ssim	0.83116	0.78439	0.82947	0.66249	0.72026	0.79729	0.72639	0.74281	0.82116	0.7653	0.89342	0.68219
time(s)	98.175	133.76	177.42	80.469	78.729	84.691	84.795	142.22	97.016	151.79	77.257	49.208
average_rsnr	9.47608333											
average_ssim	0.74487042											

- Reconstruction for the denoising network with patch-size = 512 and denoising level = 0.07

image	1	2	3	4	5	6	7	8	9	10	11	12
rsnr	10.079	7.0986	8.4008	4.8693	11.598	11.653	12.696	17.793	10.303	8.056	10.645	10.789
ssim	0.49587	0.74477	0.78404	0.35713	0.76508	0.75142	0.78818	0.92523	0.88109	0.79827	0.85668	0.8823
time(s)	207.81	46.297	191.75	200.08	114.41	237.79	237.88	173.08	87.59	238.57	240.38	104.12
image	13	14	15	16	17	18	19	20	21	22	23	24
rsnr	10.515	10.209	16.635	6.7797	8.7798	9.0974	9.0056	8.3334	17.546	6.8869	16.914	11.591
ssim	0.83469	0.79477	0.90872	0.68122	0.74266	0.83125	0.74471	0.74795	0.81063	0.7817	0.884	0.76942
time(s)	185.17	227.98	181.82	92.762	88.299	71.098	240.47	142.2	129.44	42.079	43.967	238.77
average_rsnr	10.6780625											
average_ssim	0.7734075											

- Reconstruction for the denoising network with patch-size = 512 and denoising level = 0.08

image	1	2	3	4	5	6	7	8	9	10	11	12
rsnr	9.6399	12.493	7.5772	4.4581	11.682	12.182	12.412	18.457	14.759	7.8968	9.027	11.1
ssim	0.47562	0.80183	0.68391	0.34464	0.77098	0.75858	0.79693	0.92987	0.90996	0.79369	0.8484	0.88949
time(s)	125.54	219.75	75.854	170.64	127.86	242.52	240.97	156.49	188.97	119.3	240.63	96.11
image	13	14	15	16	17	18	19	20	21	22	23	24
rsnr	10.303	11.478	18.645	7.006	10.116	10.869	10.049	9.6409	17.393	7.2595	16.763	9.0004
ssim	0.83104	0.80775	0.91754	0.67666	0.78711	0.86165	0.772	0.79418	0.80333	0.78796	0.89305	0.6607
time(s)	62.594	243.48	155.43	44.428	64.496	229.78	239.4	123.82	93.02	168.39	107.95	117.17
average_rsnr	11.2586167											
average_ssim	0.77486958											

- Reconstruction for the denoising network with patch-size = 512 and denoising level = 0.09

image	1	2	3	4	5	6	7	8	9	10	11	12
rsnr	10.185	7.3645	7.142	4.6363	11.871	11.185	19.478	18.852	6.7431	7.6554	11.065	10.003
ssim	0.4963	0.75011	0.73991	0.3434	0.78018	0.74825	0.82916	0.93348	0.85883	0.78717	0.87489	0.87072
time(s)	194.05	63.265	71.437	239.69	108.14	183.04	168.96	97.215	115.07	106.09	238.78	151.37
image	13	14	15	16	17	18	19	20	21	22	23	24
rsnr	10.495	11.204	9.0627	8.4422	9.4266	13.022	9.8784	9.1287	16.856	7.2108	15.232	12.652
ssim	0.83387	0.81302	0.86239	0.72385	0.77002	0.89435	0.77393	0.77995	0.79955	0.78362	0.87862	0.79695
time(s)	158.3	145.14	73.203	194.49	3463.2	93.625	242.55	68.817	100.5	46.311	66.536	240.52
average_rsnr	10.7829458											
average_ssim	0.780105											

- Reconstruction for the denoising network with patch-size = 512 and denoising level = 0.10

image	1	2	3	4	5	6	7	8	9	10	11	12
rsnr	9.6989	11.055	8.2795	4.444	10.913	8.2834	12.477	17.866	13.524	8.1185	8.1225	15.255
ssim	0.47754	0.75938	0.78331	0.317	0.73381	0.69461	0.77908	0.92593	0.87984	0.79004	0.79571	0.9538
time(s)	207.74	163.75	286.38	122.63	110.28	172.29	147.03	314.79	283.83	213.36	128.14	268.74
image	13	14	15	16	17	18	19	20	21	22	23	24
rsnr	7.9226	8.8671	15.395	5.9212	7.6955	13.991	8.586	7.4853	15.186	6.3609	17.398	5.4589
ssim	0.76651	0.76543	0.8878	0.59745	0.70988	0.88876	0.71635	0.72452	0.72374	0.75251	0.8854	0.54583
time(s)	130.43	154.47	134.52	16.696	248.47	314.09	226.19	82.942	22.337	112.37	89.485	31.141
average_rsnr	10.3460125											
average_ssim	0.74392625											

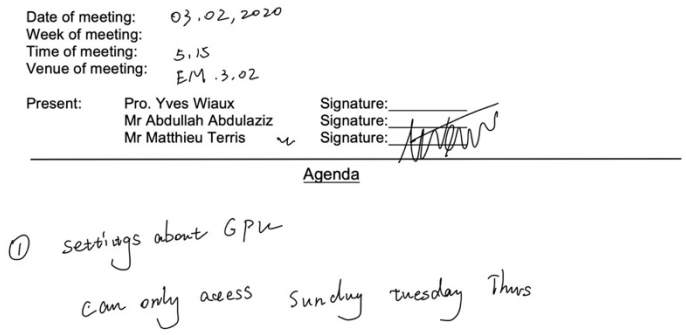
- Reconstruction for the denoising network with patch-size = 512 and denoising level = 0.13

image	1	2	3	4	5	6	7	8	9	10	11	12
rsnr	9.3656	10.407	5.8323	4.0532	10.797	10.251	6.5686	15.2	7.5625	7.2258	12.937	14.892
ssim	0.47245	0.74575	0.64072	0.31484	0.72687	0.72909	0.59069	0.89093	0.82934	0.74371	0.84293	0.96438
time(s)	143.59	232.64	44.856	245.17	91.076	242.69	41.132	106.77	204.7	56.605	226.79	108.8
image	13	14	15	16	17	18	19	20	21	22	23	24
rsnr	8.4423	10.54	15.536	6.4532	8.4995	10.074	9.2426	8.5488	17.433	6.8687	15.894	10.905
ssim	0.78785	0.79542	0.88976	0.66121	0.72459	0.80232	0.73422	0.74222	0.8087	0.7676	0.87031	0.71933
time(s)	167.7	237.5	208.06	97.533	36.731	71.235	255.5	107.86	109.27	41.713	144.74	268.94
average_rsnr	10.1470458											
average_ssim	0.74146792											

- Reconstruction for the denoising network with patch-size = 512 and denoising level = 0.16

image	1	2	3	4	5	6	7	8	9	10	11	12
rsnr	9.6551	8.2924	5.7471	4.474	10.15	5.2483	7.8752	12.306	4.279	8.4898	9.0836	11.338
ssim	0.48683	0.6514	0.61562	0.33181	0.68516	0.56728	0.61274	0.80518	0.62707	0.77569	0.75172	0.75871
time(s)	267.48	232	57.158	256.99	165.05	281.63	109.72	158.85	61.877	142.28	69.434	94.921
image	13	14	15	16	17	18	19	20	21	22	23	24
rsnr	8.0641	5.9428	13.023	19.436	5.8917	8.9578	5.7524	6.7105	17.097	4.2881	7.0761	9.6415
ssim	0.75484	0.62762	0.79253	0.84567	0.70094	0.75495	0.64572	0.72992	0.80577	0.57782	0.70652	0.69525
time(s)	274.58	260.95	210.6	247.18	193.19	130.79	57.376	258	186.95	151.92	246.65	125.45
average_rsnr	8.7008125											
average_ssim	0.67944833											

4. Minutes of the meeting



Date of meeting: 03.02.2020
Week of meeting:
Time of meeting: 5.15
Venue of meeting: EM 3.02

Present: Pro. Yves Wiaux Signature:
Mr Abdullah Abdulaziz Signature:
Mr Matthieu Terris Signature:

Agenda

① settings about GPU
can only access Sunday tuesday Thurs

meeting time : 17, 02, 2020.

meeting place : EM 3.02

meeting contents :

① talk about the process that we get the blured pic

② discuss about the M2. : use the original u-net

③ discuss about the M3.

- ADMM Algorithm
- super parameters.

Signature :