Deep Neural Networks For Inverse Problems In Imaging

Ayoub Mouhdat

60/65/65080

A thesis presented for the Master's degree of

Mathematical Sciences

Supervisors:

Giuseppe Rodriguez
Giorgio Fumera

Department of Mathematics & Computer Science

University of Cagliari, Italy

February, 2022

Abstract

Recent inverse problems research aims to develop a mathematically coherent foundation based on data-driven models. These researches use deep learning to deal with the limits in the classical methods for solving inverse problems. As a matter of fact, these classical methods are often time-consuming and are limited to solving particular problems. The present thesis investigates the aforementioned methods. We cover specifically three classical regularization methods. Moreover, we deep dive into one recent method which aims to solve the problem of image denoising using deep learning and architectures based on convolutional neural networks. This problem is considered to be one of the most famous inverse problems in imaging.

Dedications

To my beloved Mother Fatima

Acknowledgement

I would like to thank my supervisors, Prof. Giuseppe Rodriguez and Prof. Giorgio Fumera from the University of Cagliari for contributing to the success of this project with their commitment and sense of responsibility. This work would not have proceeded without their welcomes, collaboration, support, and their relevant advice regarding the organization and the project's progress.

I would also like to thank all the Department of Mathematics and Computer Science staff, professors, and employees for their immense support.

Finally, I express my gratitude to my family, friends, and all those who put me on the right track and helped me achieve my goals, including realizing this modest work.

Contents

Introduction and Motivation

1. Overview of inverse problems

From a mathematical point of view, the concept of inverse problem has a certain degree of ambiguity, which is well illustrated by J.B KELLER ‘’ Two problems are said to be inverses of each other if the formulation of one calls into question the other ‘’, a more operational definition is that an inverse problem is to determine causes with known effects. Thus this problem is the reverse of the one called direct problem, consisting of deducing the effects when the causes being known. From the definition of an inverse problem, we can see that it is likely to cause difficulties, indeed it is reasonable to demand that a direct problem be well posed : the same causes produce the same effects, on the other hand, it is easy to imagine that the same effects can come from different causes. Another difficulty in the study of inverse problems is that it often requires a good knowledge of the direct problem because these are generally situations in which we are ignorant of a set of information, some of which may be related to geometry, materials, or initial conditions. And in order to reconstruct the lost information as much as possible, it is necessary to have information perhaps partial about the output. So we try to get back to certain characteristics, usually internal and outside the direct measurement, by using information about the physical model knowing the output.

These problems fall into two large groups. On the one hand, there are the linear problems that boil down to solving an integral equation of the first type in the continuous state or to solving a linear system in the discrete case, the use of functional analysis and linear algebra provides precise results and algorithms effective. On the other hand, there are nonlinear problems that we will not address in this work.

Now, the question " what we mean by inverse problems? " may seem simple but in fact it still holds a lot of ambiguity. From an empirical point of view, we can define inverse problems as follows:

The fact of determining a non-directly observable quantity x from a finite set of measurements of an observed quantity b depending on parameters θ according to the model

$$b = A(x;\theta) \tag{1}$$

This equation can be considered as the main objective to solve a set of real problems in different experimental sciences. we can say that:

- The calculation of b given A, θ, x : is a **direct problem**.
- The calculation of x given A, θ , b : is an **inverse problem**.

So we can classify as an inverse problem every situation in which we wish to evaluate a particular physical quantity x that is inaccessible to the experiment by the measurement of another quantity b that is directly accessible to the experiment. Knowing a mathematical model of the direct problem which explicitly gives b from x. Having doubts in the model and measurements greatly affects this kind of problem, so it is more realistic to written:

$$b = A(x,\theta) + e \tag{2}$$

where e represents the errors commonly called noise.

A typical property of inverse problems is ill-posedness, a property which is opposite of well-posedness that was introduced by *J.HADAMARD* (1) from a well-posed :

- There exist a solution ;
- The solution is unique ;

- The solution is continually dependent on the data.

A problem that does not satisfy one of the conditions mentioned above is ill-posed, the inverse problems often do not satisfy one of these conditions, or even all three together. This is not surprising for several reasons First, a physical model is fixing, the available empirical data is usually noisy, and there is no guarantee that this noisy data comes from this model. Let us analyze the three conditions of a well-posed problem in the case of inverse problems.
First the existence condition seems to be trivial since we can easily formulate problems that do not have a solution. However the fact that the solution to an inverse problem may not exist is not a great difficulty. It is usually possible to restore existence by relaxing the notion of solution.
Second non-uniqueness is a little more serious, if a problem has more than one solution we need additional information (prior information). To choose between them. The stability condition is much harder to deal with because a violation implies that arbitrarily small perturbations of data can produce arbitrarily large perturbations of the solution.

The key is to reformulate the problem such that the solution to the new problem is less sensitive to the perturbation, we say that we stabilize or regularize the problem. Regularization is the heart of all our discussion, it is the process of adding information in order to solve an ill-posed problem or to prevent over-fitting, and can be applied to objective functions in ill-posed optimization problems. The regularization term or penalty imposes a cost on the optimization function to make the optimal solution unique. In general, we could think about the regularization in inverse problems like this :

$$\hat{\mathrm{x}} = \mathrm{Argmin}\, \|\mathrm{b} - \mathrm{Ax}\|_2^2 + \mathrm{r}(\mathrm{x}) \tag{3}$$

We have got some observations b and we're going to try to estimate x by perhaps solving an optimization problem where we try to minimize a measure of error such as squared error plus some regularizer r, and the question is what this regularizer should be. We focus on the discrete inverse problem defined by a system of linear algebraic equations:

$$b = Ax + e \tag{4}$$

It is obvious that if we want to solve $Ax = b$, and A is a regular square $(\det A \neq 0)$, then the unique solution is simple and worthy. this can be calculated in different numerical ways. On the other hand, if we want to solve $Ax = b$ with $A \in R^{m \times n}$, and if we assume that the problem is ill-posed, there are several approaches according to Hadamard's condition which are not respected. We focus on the regularization methods.

The matrix A in discrete ill-posed problem is often highly ill-conditioned, and when this is the case it is well known from matrix computations that small perturbations of b can lead to large perturbations of the solution.

Such difficulties and others is always face us whenever we solve an inverse problem, and before addressing these difficulties and present some strategy to regularize an ill-posed problem we have to define some building blocks:

- $\|\mathrm{b} - \mathrm{Ax}\|_2$: we call it a data fidelity term, measures the goodness-of-fit (how well the solution predicts the given data b.
- $\|\mathrm{x}\|_2$: we call it a prior knowledge because the incorporation of this term is based on our knowledge that the naïve solution is dominated by high-frequency components, and the hope is therefore that if we control the norm of x then we can suppress the large noise components.

We are kind of having multiple stages of enlightenment, our work is going to be based on:

1. Presenting classical methods of regularization for solving inverse problem.
2. Using deep neural network on solving an inverse problem in imaging.

2. Examples

Image reconstruction (deconvolution)

Inherently, all image generation systems have a resolution limit. Furthermore, due to a number of physical phenomena such as motion in the object or picture planes, turbulence, light diffusion, and other physical effects, some images can be blurred and/or distorted. When an image has been deteriorated in this way, a variety of digital image processing techniques can be used to 'de-blur' it and improve its information richness. As a result, 'de-blurring' an image entails solving the model's inverse problem, known as Image reconstruction(deconvolution), is an inverse problem that deals with the reconstruction of data from known sources (2). It is vitally dependent on past understanding of how the data (the digital image) was generated and recorded, this inverse problem occurs When the pixel value is affected by nearby pixels. Deconvolution attempts to produce a resolution that is suitable with the imaging system's bandwidth (a resolution limited system). The goal of image reconstruction is to give a resolution that is higher than the data's inherent resolution (i.e. the resolution limit of the imaging system). We will discuss this in detail in the second part. The data obtained is usually mathematically connected to an object function via an integral transform. Deconvolution is concerned with inverting certain types of integral equations, notably the convolution integral, in this way. Because it is an ill-posed problem, there is no exact or unique solution to the image reconstruction problem.

Sar imaging inverse problems

Synthetic aperture radar (SAR) is a critical remote sensing technique that can provide high-resolution photographs of the Earth at any time of day or night (3), in a variety of terrains, and in difficult situations. Despite the fact that SAR photos have resolutions as high as, factors such as atmospheric delays and speckle noise still limit the quality of SAR photographs. As a result, it's critical to improve this in order to make target detection and tracking, classification, and security-related jobs easier. In all imaging systems, the difficulty of estimating an object of interest directly from measurements (image) arises. Imaging inverse problems are a general term for situations of this nature. SAR imaging inverse problems are one example of this type of challenge. One of them, as we saw in the previous example, is image reconstruction. We can also find Ship Wake Detection in the inverse problems of Sar imaging. A moving ship in the deep sea produces various types of wakes, one of which is turbulent wake, which limits the moving ship's signatures. Ship wake detection approaches mainly require solving an inverse problem, assuming that ship wakes can be described as linear structures.

Chapter 1

Classical regularization methods for ill-posed problems and their applications

In the following we present different methods for computing approximate solutions that are less sensitive to perturbations. These methods are called regularization methods since they impose regularity on the computed solution usually in the form of condition that the solution is smooth. We suppress some unwanted noise components, resulting in more stable approximate solutions.

1. The Singular Value Decomposition

The Singular Value Decomposition is a very powerful tool for analyzing discrete ill-posed problems, for any matrix $A \in R^{m\times n}$ with $m \geq n$, the SVD takes the form :

$$A = U\ \Sigma\ V^{\top} = \sum_{i=1}^{n} u_i\ \sigma_i\ v_i^{\top} \tag{1.1}$$

- $\Sigma \in R^{n\times n}$ is a diagonal matrix with the singular values, satisfying : $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$

- $U \in R^{m\times n}$ and $V \in R^{n\times n}$ consist of the left and right singular vectors and both Matrices have orthonormal columns i.e $U^\top U = V^\top V = I$

The singular value and vectors obey a number of relations, the most important of them are:

$$Av_1 = \sigma_1 u_t, \quad \|Av_1\|_2 = \sigma_1, \quad i = 1, \ldots, n \tag{1.2}$$

if A is square and nonsingular we can found the second relation:

$$A^{-1}u_1 = \sigma_1^{-1} v_t, \quad \left\|A^{-1}u_1\right\|_2 = \sigma_1^{-1}, \quad i = 1, \ldots, n \tag{1.3}$$

We can use some of these relations to find an expression for the solution x (1). The matrix V is orthogonal so we can always write the vector x in the form

$$x = VV^\top x = V \begin{pmatrix} v_1^T x \\ \vdots \\ v_n^T x \end{pmatrix} = \sum_{i=1}^{n} \left(v_1^T x\right) v_i \tag{1.4}$$

Similarly the matrix U is orthogonal we can also write the vector b in the form

$$b = \sum_{i=1}^{n} \left(u_i^\top b\right) u_i \tag{1.5}$$

Now we combine the expression for x with the SVD for A we obtain:

$$Ax = \sum_{i=1}^{n} \sigma_i \ \left(v_i^\top x\right) \ u_i \tag{1.6}$$

By comparing the coefficients in the expression $Ax = b$ we see immediately that the solution is given by:

$$x = A^{-1}b = \sum_{i=1}^{n} \frac{u_1^T b}{\sigma_t} v_r \tag{1.7}$$

In connection with discrete ill-posed problems, at the SVD of A we often find two characteristic features, those we found in many discrete ill-posed problems arising in practical applications. The first one, is that the singular values σ_i decay gradually to zero with no particular gap in the spectrum, and will increase the number of small singular values by increasing the dimensions of A. The second is that the singular vectors u_i and v_i tend to

cause more changes in their elements as the index i increases, i.e., the lower σ_i.
The SVD gives important insight into an aspect of discrete ill-posed problems, namely, the smoothing effect , as we have seen when σ_i decreases, the singular vectors u_i and v_i become more and more oscillatory. Consider now the system $Ax = b$ of an arbitrary vector x. Using the SVD we get:

$$x = \sum_{i=1}^{n} \left(v_i^{\top} x\right) v_i \tag{1.8}$$

and

$$Ax = \sum_{i=1}^{n} \sigma_i \ \left(v_i^{\top} x\right) \ u_i \tag{1.9}$$

This clearly shows that the due to the multiplication with the σ_i the high-frequency components of x are more damped in Ax than low-frequency components. Moreover, the inverse problem namely that of computing x from $Ax = b$, must have the opposite effect it amplifies the high-frequency oscillations in the right-hand side b.
In order to understand how the small singular values affect the solution we consider a simple model:

$$b = Ax + e \tag{1.10}$$

Where the vector x represents the true and exact solution and e is the noise that is always present in data, and b is the measured data. These quantities are unknown in practice but our aim is to find a vector approximating to x .
By substituting the singular-value decomposition of A we obtain:

$$\boldsymbol{b} = \left(\sum_{i=1}^{n} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T\right) \boldsymbol{x} + \boldsymbol{e} \tag{1.11}$$

As we have seen the approximate solution is given by :

$$x = A^{-1} b = \sum_{i=1}^{n} \frac{u_1^T b}{\sigma_t} v_r \tag{1.12}$$

We can see that the relationship between the exact and approximate solution is as follows:

$$\tilde{\boldsymbol{x}} = \sum_{i=1}^{n} \left(\boldsymbol{v}_i^T \boldsymbol{x} + \frac{\boldsymbol{u}_i^T \boldsymbol{e}}{\sigma_i} \right) \boldsymbol{v}_i = \boldsymbol{x} + \sum_{i=1}^{n} \frac{\left(\boldsymbol{u}_i^T \boldsymbol{e} \right)}{\sigma_i} \boldsymbol{v}_i \tag{1.13}$$

We see that the error term is divided by the singular value σ_i, then if some single values of it are small which is the case in most problems in reality, this division will give a very large random component, often completely overwhelming the x component. Therefore, as soon as there is noise even if it is small we will lose the exact solution completely.
Then the least squares method usually fails when small single values are present. It is better to consider small single values effectively as zero, this is what we will introduce in the following, and is known as Truncated SVD and Selective SVD.

2. Methods for the repair of SVD

2.1. Truncated SVD

The extremely large errors in the solution that we computes by SVD (naïve solution), come from the noisy SVD components associated with the smaller singular values. The idea behind TSVD is to chop those SVD components that are dominated by the noise, so by retaining the first k components of the naïve solution.

The TSVD regularization method is based on this observation where solving the problem: min $\|\mathrm{b} - \mathrm{Ax}\|_2$ is needed in order to find the solution, but it is not enough to give us a unique and regular solution , we do that by a prior knowledge $\|\mathrm{x}\|_2$. The truncated SVD solution x_k corporate these to the requirement and it is defined by:

$$x_k \equiv \sum_{i=1}^{k} \frac{u_i^\top b}{\sigma_i} v_i \tag{1.14}$$

The truncation parameter k should be chosen such that all the noise-dominated SVD coefficients are discarded (4).

There is an alternative version of the TSVD method. No less important than the first, while the definition of x_k takes its basis for calculating the regularized solution in a specific formula. We can define a regularized solution as the solution to a modified and better conditioned problem. Let us introduce the TSVD matrix A_k, which is the $\text{rank}-k$ matrix defined as:

$$A_k = \begin{pmatrix} | & & | \\ u_1 & \cdots & u_k \\ | & & | \end{pmatrix} \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{pmatrix} \begin{pmatrix} | & & | \\ v_1 & \cdots & v_k \\ | & & | \end{pmatrix}^{\mathsf{T}} = \sum_{i=1}^{k} u_i \ \sigma_i \ v_i^{\mathsf{T}} \tag{1.15}$$

We have seen that the singular values of A satisfies $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. It can be shown that the condition number of this matrix is $\text{cond}(A_k) = \sigma_1/\sigma_k$, and it is immediately much smaller than the condition number $\text{cond}(A) = \sigma_1/\sigma_n$ of the original matrix A. Hence a good idea is to replace the original and ill-conditioned problem $Ax = b$ with the better conditioned problem $A_k x = b$. In this case the least squares formulation $\|\mathrm{b} - \mathrm{A_k x}\|_2$ is needed, because we can no longer expect to find an x such that $A_k x = b$. However, since A_k is rank deficient (as long as $k < n$), there is not a unique solution to this least squares problem, to determine a unique solution, we should add a constraint to the least square problem. and this constraint is generally seeking the solution in many applications with minimum 2-norm i.e:

$$\min \|x\|_2 \quad \text{subject to} \quad \min \left\|A_k x - b\right\|_2 \tag{1.16}$$

It is clear to show that the solution to this constraining problem is exactly the TSVD solution x_k, and thus we can consider it an alternative definition of the TSVD solution, in which a minimization of its modulus $\|\mathrm{x}\|_2$ is the regularity requirement on x.

The fact of based the criterion for choosing the truncation parameter k on the size of the singular values σ_i, it is a fairly common mistake in connection with TSVD. The truncation parameter k should define the break-point between the retained and discarded (filtered) SVD coefficients (4), as long as the noise is restricted to the right-hand side, and therefore the choice of k in this case is determined based of the noisy coefficients $U_i^T b$.

2.2. Selective SVD

The rationale behind the TSVD regularization and many others is that the underlying problem satisfies the Picard condition (1) . Hence we are assured that regular solutions will capture important components of the exact solution, TSVD depends on including all SVD components corresponding to the largest single values but this is not always necessary, for example we can consider a problem where we say that every second component of the SVD is zero, in In this case the inclusion of these SVD components is meaningless.

We introduce a variant of the TSVD method, this variant is called SSVD method in which we include or select only the components of the SVD that make them important contributions to the structured solution. Specifically, given a threshold τ for the right-hand side's SVD coefficients. The SSVD solution x_τ is defined as:

$$x_\tau \equiv \sum_{|u_i^\top b|>\tau} \frac{u_i^\top b}{\sigma_i} v_i \tag{1.17}$$

We sum all the SVD components $\left(U_i^T b\right)/\sigma(\text{i})$, for which the absolute value $\left|U_i^T b\right|$ of the right-hand side's SVD coefficient is above the threshold t. Thus the filter factors for the SSVD method are:

$$\varphi_1^{[\tau]} = \begin{cases} 1, & \left|u_1^\top b\right| \geq \tau \\ 0, & \text{otherwise} \end{cases} \tag{1.18}$$

It is natural to choose the threshold τ such that the SSVD filters remove the coefficients $U_i^T b$ below their noise level. We know that the noise in the right-hand side b is white then the noise in the coefficients $U_i^T b$ is also white with the same statistics Thus we can choose t in order to avoid including components at the noise level.

3. Tikhonov Regularization

The TSVD method has a large set of advantages, the most important of which is that it is intuitive. Once computed the SVD , TSVD solutions x_k can easily be calculated for different truncation parameters, but in contrast, it has a clear drawback, which is that it explicitly requires the calculation of SVD or, at least, the k singular values and vectors. This computational task can often be very overwhelming when it comes to solving big problems (5), and therefore there is a need for other methods of Regularization that are more suitable for this type of problems.

Tikhonov is one of successful regularization method of all time. Similarly, with the TSVD, tikhonov's method explicitly incorporates the regularity requirement in the reformulation of the problem, which it is replaced by another one will-posed, but the difference between them is that the *Tikhonov* regularization is based on the idea that we want to minimize a combination of data fidelity and a prior knowledge. We don't give the same importance to both through the solution of this problem:

$$\min_x \left\{ \|Ax - b\|_2^2 + \lambda^2 \|x\|_2^2 \right\} \tag{1.19}$$

The main difficulty in the application of *Tikhonov* regularization in particular problems is the determination of the regulation positive parameter λ.

Through the factor λ^2. we control the balance between the two terms, If λ is larger we mainly focus on the solution criterion $\|\mathrm{x}\|_2$ (a prior knowledge) to minimize the combination, i.e., more weight is given to reduce $\|\mathrm{x}\|_2$ Because in this case, any perturbation will be very impactive. While if λ is smaller we focus on data fidelity $(\|\mathrm{b} - \mathrm{Ax}\|_2)$ with the same reasoning. Finding a good balance between these two terms depends mainly on finding a suitable value of λ.

We can use the SVD to obtain more insight into the Tikhonov solution x_λ, when we insert the SVD of A into the normal equations we obtain:

$$\begin{aligned} x_\lambda &= \left(V\Sigma^2 V^T + \lambda^2 V V^T \right)^{-1} V \Sigma U^\top b \\ &= V \left(\Sigma^2 + \lambda^2 I \right)^{-1} V^\top V \Sigma U^T b \\ &= V \left(\Sigma^2 + \lambda^2 I \right)^{-1} \Sigma U^\top b \end{aligned} \tag{1.20}$$

Which lead to the following expression:

$$x_\lambda = V \left(\Sigma^2 + \lambda^2 I \right)^{-1} \Sigma U^\top b \tag{1.21}$$

If we insert the singular values and vectors, we obtain :

$$x_\lambda = \sum_{i=1}^{n} \varphi_i^{[\lambda]} \frac{u_i^T b}{\sigma_i} v_i \tag{1.22}$$

where $\varphi_i^{[\lambda]}$ is the filter factor which satisfy :

$$\varphi_i^{[\lambda]} = \frac{\sigma_i^2}{\sigma_i^2 + \lambda^2} \approx \begin{cases} 1, & \sigma_i \gg \lambda \\ \sigma_i^2/\lambda^2, & \sigma_i \ll \lambda \end{cases} \tag{1.23}$$

We see that for singular values σ_i larger than the parameter λ, the filter factors are close to one then the corresponding SVD components contribute to x_λ with almost full strength. On the other hand, for the singular values much smaller than λ the filter factors are small, and therefore these SVD components are damped or filtred. We conclude from this SVD analysis that Tikhonov's solution is a filtered solution, in the same way as the TSVD solution.

4. Choice of the regularization parameter

We have covered three regularization approaches, but the most important item we have left out is an effective way for determining the regularization parameter.
Because it's the only way to set up an effective regularization method . As previously stated, a suitable regularization parameter should result in a reasonable balance of perturbation and regularization errors in the regularized solution. Several parameter choice procedures have been proposed. Parameter choice methods can be classified according to the input used to make the choice. There are two basic types :

1. Methods based on knowledge, or a good estimate, of $\|e\|_2$. Such methods are not discussed here

2. Methods that do not require $\|e\|_2$, but instead seek to extract the necessary information from the given right-hand side.

We present one of recent ,parameter choice method that belong to class 2 :

4.1. The l-curve criterion

For a continuous regularization parameter λ we compute the curvature of the curve:

$$\left(\log \|Ax_\lambda - b\|_2 \, , \log \|Lx_\lambda\|_2\right) \tag{1.24}$$

The *l-curve corner* is defined as a point with maximum curvature, then we choose a value of λ that corresponds to this corner.
The *2-norm* is not always the best approach to gauge the size of a solution or residual vector. The choice of regularization method determines the most natural approach to measure size. We can see that the l-curve is a parametric plot of the magnitude of the regularized solution as well as the related residual. The basic notion is that, in addition to using information about the residual size, a reasonable technique for determining the regularization parameter for discrete ill-posed problems should include information about the solution size. This is entirely natural.This is entirely logical, as we are attempting to strike a reasonable balance in keeping both of these values low. The l-curve has a characteristic l-shaped corner that marks the point at which the solution x shifts from being dominated by regularization errors to being dominated by errors on the right side. hence the l-corner curve's corresponds to a good balance, and the accompanying regularization parameter is also a good one.

When the regularization parameter is discrete, we use a 2D spline curve to approximate the discrete lcurve in log-log scale, compute the point on the spline curve with the most curvature, and define the discrete L-corner curve's as the point closest to the spline curve's corner.
There are numerous advantages to using the l-curve approach to select the regularization parameter. The computation of the comer is a well-defined numerical problem, and coupled mistakes seldom "trick" the approach. Even highly correlated errors will make the size of the solution grow once the regularization parameter A becomes too small, thus producing a corner on the L-curve.

4.2. Generalized cross-validation (GCV)

The generalized cross-validation is one of the most important ways to estimate values for regularization parameters such as k in the truncated singular value decomposition, and λ in Tikhonov, is based on the philosophy that the choice of regularization parameter should be independent of an orthogonal transformation of b, and if an arbitrary element bi of the right-hand side b is left out, then the corresponding regularized solution x^* should predict this observation well, by choosing the regularization parameter which minimizes the GCV function:

$$\mathrm{G} = \frac{\|Ax^* - b\|_2^2}{T^2} \tag{1.25}$$

Note that G is defined for both continuous and discrete regularization parameters. It can be used in both methods TSVD and *Tikhonov.* Where the numerator is the squared residual norm and the denominator is a squared effective number of degrees of freedom (4) the effective number of degrees of freedom T (which is not necessarily an integer) can be written in terms of the filter factors as:

$$T = m - \sum_{i=1}^{\mathrm{rank}(A)} f_i \tag{1.26}$$

Then we can see that In the case of :

1. *Tikhonov* regularization, GSV works as follow

$$\text{Choose } \lambda \text{ as the minimizer of} \quad G(\lambda) = \frac{\|Ax_\lambda - b\|_2^2}{\left(m - \sum_{l=1}^{n} \varphi_1^{[\lambda]}\right)^2}$$

2. TSVD regularization, GSV is much simpler :

$$\text{Choose } k \text{ as the minimizer of} \quad G(k) = \frac{\|Ax_k - b\|_2^2}{(m-k)^2}.$$

The most appealing advantage of the method is that it does not require the knowledge of the noise variance. The basic idea of GCV is that the parameter is a good choice, if it minimize the GCV function.

Chapter 2

Fundamental of deep learning

1. From the Artificial intelligence to the Deep learning

To talk about deep learning, we need to clarify what is meant by artificial intelligence and machine learning, and what is the location of deep learning in between as we can see with the help of the Figure 2.1:

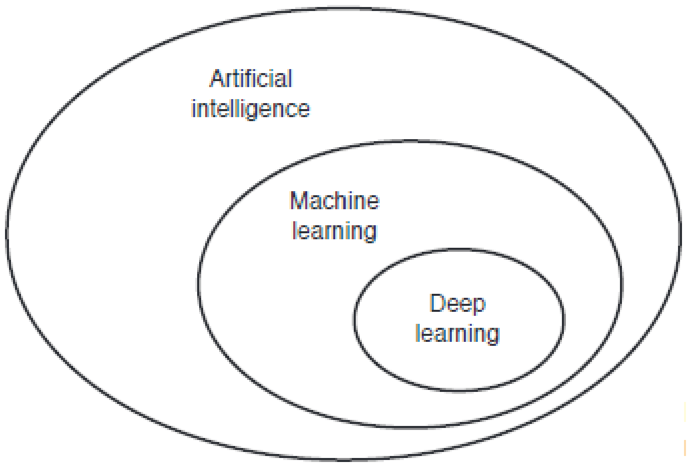


Figure 2.1: Deep learning is a subset of machine learning which in turn is a subset of AI that focuses on a narrow range of activities (6)

1.1. Artificial intelligence

Artificial intelligence (AI) began in the 1950s when a group of pioneers in the field of computer science wondered if computers might be programmed to "think," a topic whose repercussions we're still debating today. It covers a range of disciplines that have been the subject of a tremendous amount of research, scrutiny, confusion, and fanciful hype (6). It is based on a general class of algorithms capable of very effectively approximating complex nonlinear processes, which are used to automate tasks previously restricted to humans. Yet it is disingenuous to assert that today's machines are learning to "think" in any human sense of the word.
The quest to automate intellectual functions traditionally performed by humans is a succinct definition of the field. As a result, AI is a broad term that embraces not only machine learning and deep learning but also a wide range of non-learning approaches. Many scientists believed for a long time that human-level AI could be reached by having programmers construct a large enough set of explicit rules for manipulating knowledge. From the 1950s through the late 1980s this technique was known as symbolic AI and it was the dominant paradigm in AI.

Although symbolic AI proved to be suitable for handling well-defined, logical problems such as chess. figuring out explicit rules for more complex, fuzzy issues such as picture classification, speech recognition, and language translation proved to be intractable. A new approach has emerged to take the place of symbolic AI and that is machine learning.

1.2. Machine learning

Machine learning originates from the question of whether a machine can learn on its own how to accomplish a task beyond what humans know how to instruct it to do. Is it possible that a computer will surprise us? Could a machine learn data-processing rules by looking at data instead of programmers constructing them by hand?

This question ushers in a new paradigm of programming. Humans input rules (a program) and data to be processed according to these rules in classical programming, which is the paradigm of symbolic AI. Humans input data as well as the expected replies from the data, and machine learning generates rules. These rules can then be applied to new data to provide unique responses.

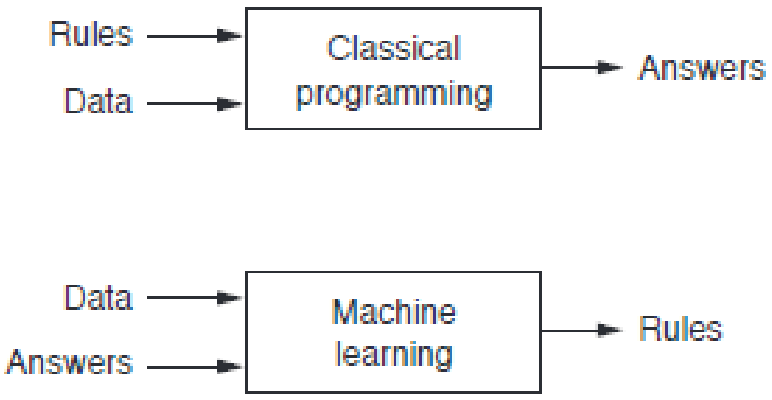


Figure 2.2: Replacing the classic programming paradigm with machine learning (6).

Rather than being explicitly designed, a machine-learning system is trained. It is presented with a large number of instances related to a task, and it detects statistical structure in these examples, allowing the system to develop rules for automating the work. Although machine learning is closely related to mathematical statistics, it differs from statistics in a number of ways. Machine learning, unlike statistics, deals with enormous, complicated datasets (such as a dataset of millions of photos, each with tens of thousands of pixels) for which traditional statistical analysis, such as Bayesian analysis, is impracticable.

To explain deep learning and understand how it differs from previous machine-learning approaches, we must first grasp what machine learning algorithms are. Machine learning as we have just stated develops rules to carry out a data-processing task given instances of what is expected. So we'll need three things to do in machine learning :

1. Input data points : As an example, they could be photos if the task is image tagging.
2. The expected output : "dog," "cat," and so on could be predicted in an image task.
3. A measure for testing whether the algorithm is doing a good job : This is required in order to calculate the distance between the algorithm's current output and the target output.

A machine-learning model learns how to transform its input data into meaningful outputs by being exposed to known examples of inputs and outputs. As a result, the primary problem in machine learning and deep learning is to convert data in a meaningful way, we want to learn valuable representations of the input data we have on hand-representations that will help us move closer to the predicted output. Before we go any further, let's define what a representation is. It is at its root a new way of looking at data—a new way to describe or encode data. For example, a color image might be encoded in the RGB (red-green-blue) or HSV (hue-saturation-value) formats which are two distinct representations of the same data. Some tasks that are tough with one representation become simple with a different one. For example, in the RG format the process of "selecting all red pixels in the image" is easier whereas in the HSV format, the effort of "making the image less saturated" is easier. We have to Found appropriate representations for their input data-transformations of the data that make it more accessible to the task at hand.

So, to put it another way machine learning is the process of looking for meaningful representations of some input data within a preset field of possibilities while being guided by a feedback signal. This simple concept can be used to solve a wide range of problems.

1.3. Deep learning

Deep learning is a subfield of machine learning that emphasizes learning successive layers of increasingly meaningful representations when learning representations from data. The deep in deep learning refers to the idea of multiple layers of representations rather than

any form of deeper knowledge produced. The depth of the model refers to how many layers contribute to a data model. Layered representations learning and hierarchical representations learning could have been better names for the field. Tens or even hundreds of consecutive layers of representations are typically used in modern deep learning, and they're all learned automatically from training data. these layered representations are (nearly always) learned in deep learning using neural network models, which are arranged in literal layers piled on top of each other (7). Although some of the key principles in deep learning were developed in part by drawing inspiration from our understanding of the brain, deep-learning models are not models of the brain, as the term neural network implies. There is no indication that the brain uses learning techniques similar to those used in recent deep-learning models.
In short, deep learning is a mathematical framework for learning representations from data for our purposes. To understand it, you need to be familiar with a number of concepts. When we discuss deep learning, we generally refer to deep neural networks.

2. Deep neural networks

2.1. Definition and components

Deep neural networks have recently become the standard tool for solving a variety of computer vision problems. we will look at the elements that make it up and that are also used to train it:
Deep neural networks are a system as shown in the Figure 2.3. that is composed of small Nodes that function similarly to neurons in the human brain. When they are stimulated a reaction occurs in these nodes, some are connected and marked, while others are not although nodes are generally grouped into layers (8). the system must process layers of data between the input and output to solve a task.

Layers: The layer is the most basic data structure in neural networks. Is a data processing module that accepts one or more tensors as input and produces one or more

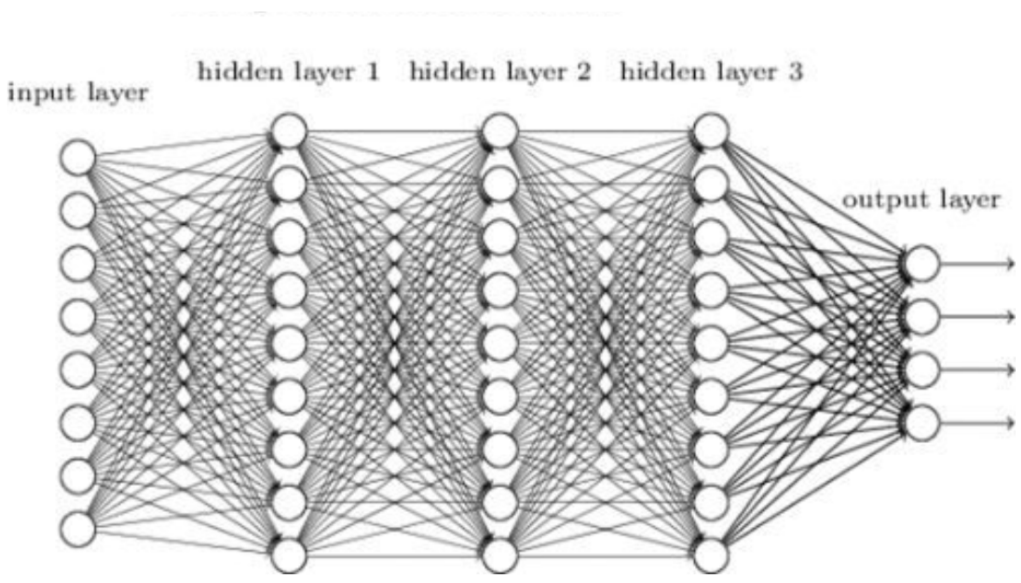


Figure 2.3: Deep neural network architecture (7).

tensors as output. Some layers are stateless, but the majority of them have a state: the layer's weights, one or more tensors learned using stochastic gradient descent that They hold the knowledge of the network as a whole. Different layers are acceptable for various tensor formats and data processing method. For example, densely linked layers also known as fully connected or dense layers, are frequently used to handle simple vector data stored in 2D tensors of shape. Recurrent layers such as an LSTM layer, often handle sequence data which is stored in 3D tensors of shape. 2D convolution layers are typically used to process image data contained in 4D tensors.

Model (networks of layers): A directed acyclic graph of layers is a deep-learning model. A linear stack of layers which maps a single input to a single output is the most typical example. There is however a wide variety of network architectures, a hypothesis space is defined by the architecture of a network, we defined machine learning as "looking for usable representations of some input data within a predefined range of possibilities, guided by a feedback signal." by deciding on a network architecture, you limit your range of options (hypothesis space) to a set of tensor operations, that map input data to output data. Then you'll need to find a good set of values for the weight tensors that are involved in these tensor operations.

Loss functions and optimizers: After we have defined the network structure, we will need to choose two more things: first, the loss function (objective function) which determines how much will be lowered during training. It is a metric for determining whether or not the task at hand was completed successfully. The second component, is an optimizer which determines how the network should be updated based on the loss function.

Choosing the proper objective function for the right problem is critical, our network will take every shortcut it can to reduce loss, so if the objective does not entirely correlate with success for the task at hand, our network may do things we may not want.

2.2. How deep neural networks works ?

At this point, we know that machine learning is concerned with mapping inputs (such as noisy images) to targets (such as "clean image"), which is accomplished by seeing a large number of pairs of input and target observations. We have also shown that deep neural networks perform this input-to-target mapping using a deep sequence of simple data transformations (layers) which learns from examples. We show how this learning takes place in practice.

The weights of a layer which are essentially a set of numbers, store the definition of what a layer performs with its incoming data. In technical terms, we can state that a layer's transformation is parameterized by its weights (see figure 2.4). (A layer's weights are also known as its parameters). Learning in this instance refers to determining a set of weights for all layers in a network that allows the network to accurately map sample inputs to their corresponding targets.

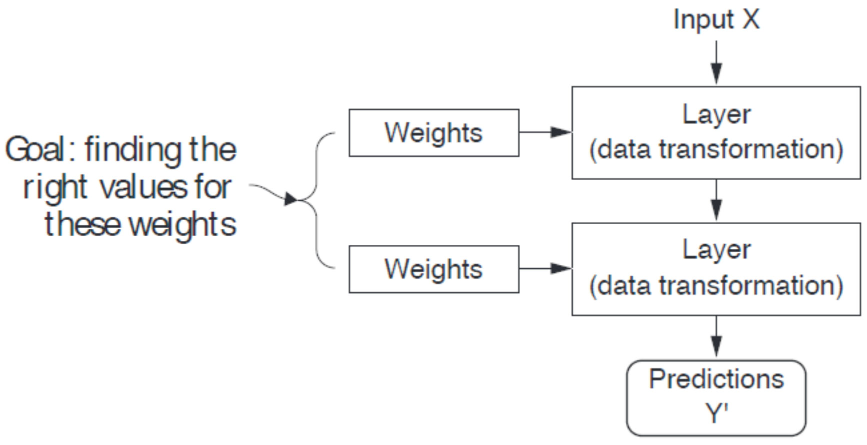

Input X
Goal: finding the right values for these weights
Weights
Layer (data transformation)
Weights
Layer (data transformation)
Predictions Y'

Figure 2.4: The weights of the neural network are the main parameters.

Next, comes the role of the loss function for the network, which is based on measuring how far this result is from what you expected. It takes the network predictions and the real target, calculates the distance score and capture How well the network performs in this specific example (see figure 2.5). The essential idea in deep learning is to utilize this score as a feedback signal to adjust the weights' values slightly, in the direction of lowering the current example's loss score (see figure). This adjustment is the job of the optimizer, which implements what's called *Backpropagation* algorithm which represent the central algorithm in deep learning.

The network's weights are initially given random values, so it just performs a series of random transformations. Naturally, its productivity falls well short of what it should be, and the loss score reflects this. However, as the network analyzes more examples, the weights are modified somewhat in the right direction, and the loss score lowers. A network with a minimal loss is one for which the outputs are as close as they can be to the targets (trained network).

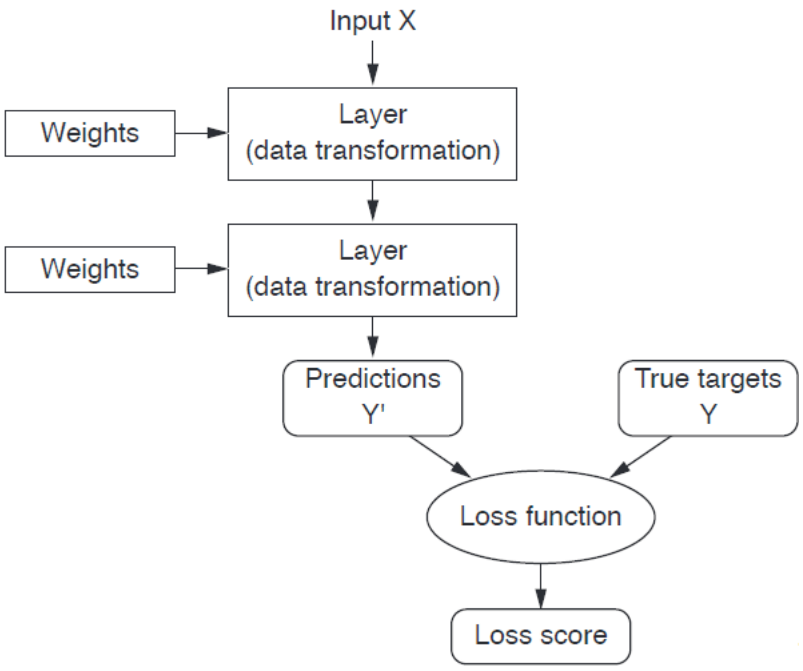


Figure 2.5: Evaluate the network output quality using loss function.

3. Convolutional neural networks

There are many models of deep learning for different tasks that we are trying to solve, as we have seen each model is ideal for a specific task. Our main task in this work is the solving inverse problems in imagine. Therefore, we introduce convolutional neural networks also known as convnets, a type of deep learning model that is often used in image recognition that we are interested as an application.

CNN is a type of deep learning model for processing data with a grid pattern such as images. This class of artificial neural networks it has been a dominant method in computer vision tasks, designed to automatically and adaptively learn spatial hierarchies of features, from low- to high-level patterns. Convolutional neural networks (CNNs) are generally made up of three types of layers: convolution, pooling and fully connected layers.

The convolutional layer is the most important layer in a CNN because it is where

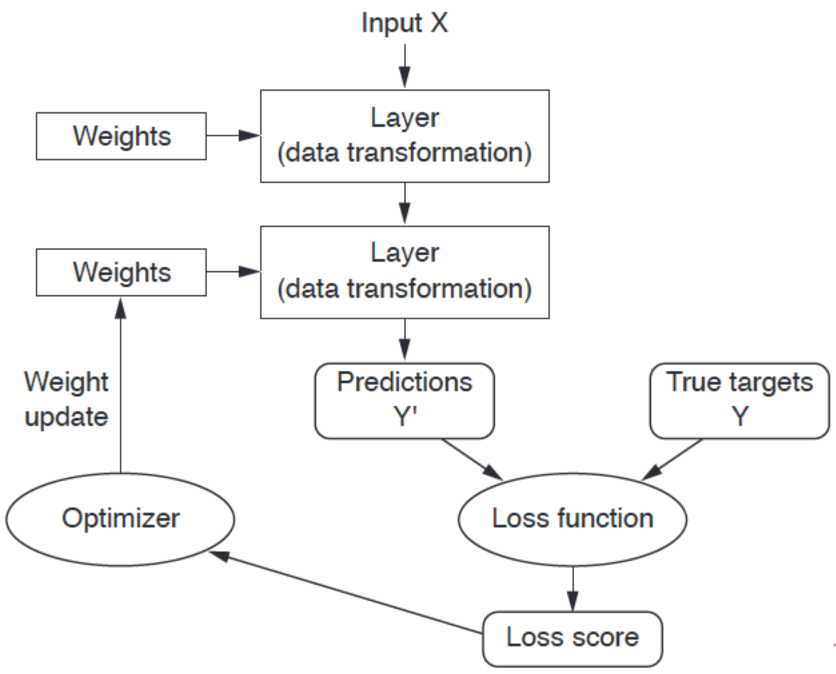


Figure 2.6: How weights are adjusted when training the network using the loss score.

the majority of computation takes place (10). It requires input data, a filter and a feature map among other things. Let's pretend the input is a color image which is made up of a 3D matrix of pixels. This means the input will have three dimensions: height, width and depth which match to the RGB color space of a picture. A feature detector also known as a kernel or a filter will traverse across the image's receptive fields, checking for the presence of the feature. Convolution is the term for this procedure. The feature detector is a two-dimensional (2-D) weighted array that represents a portion of the image. The filter size which can vary in size, is usually a 3x3 matrix which also affects the size of the receptive field. The filter is then applied to a portion of the image, and the dot product between the input pixels and the filter is determined. After that, the dot product is loaded into an output array. The filter then shifts by a stride, and the procedure is repeated until the kernel has swept across the entire image.

As you can see in the image below, it is not necessary to connect every output value in

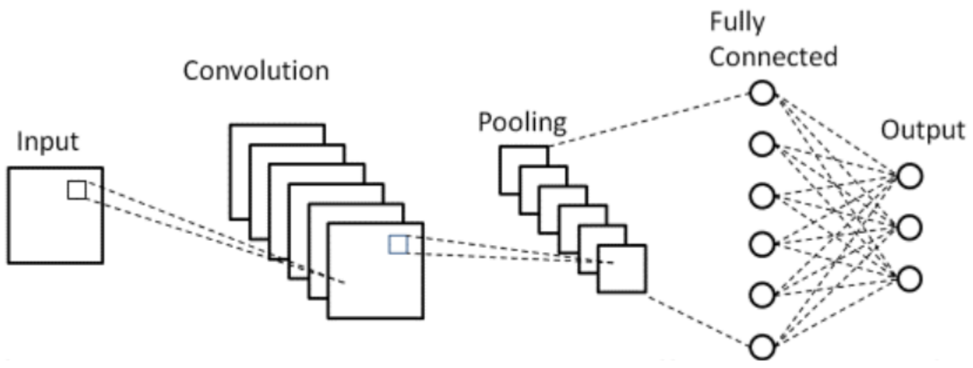


Figure 2.7: An example of CNN architectures for a classification task (9)

the feature map to every pixel value in the input image.

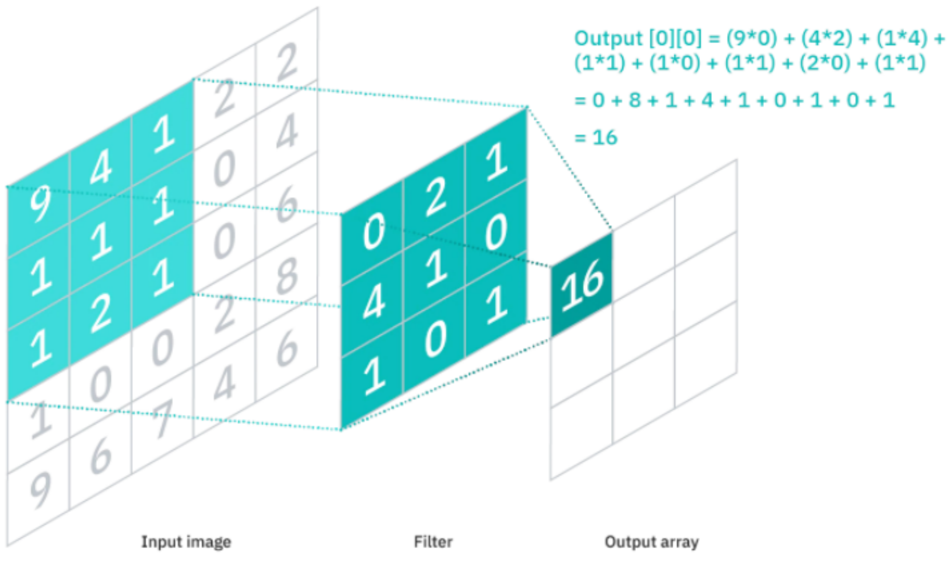

Output [0][0] = (9*0) + (4*2) + (1*4) + (1*1) + (1*0) + (1*1) + (2*0) + (1*1)
= 0 + 8 + 1 + 4 + 1 + 0 + 1 + 0 + 1
= 16
Input image
Filter
Output array

Figure 2.8: The filter is applied by the convolution layer (9)

Before the neural network training begins, there are three hyper-parameters that determine the output volume size that must be established. These are some of them:

- **Number of filters** : has an impact on the output's depth. Three distinct filters, for example, would result in three different feature maps, resulting in a depth of three.

- **Stride** : is the distance, in pixels, that the kernel travels across the input matrix. Although stride values of two or more are uncommon, a bigger stride results in a lesser output.
- **Padding** : When the filters don't fit the input image, it's frequently utilized. This reduces the size of all elements outside of the input matrix to zero, resulting in a larger or more evenly proportioned output.

We now know that convolution is the process of multiplying and adding pixel values by weights (hence the name Convolutional Neural Network). A CNN usually consists of several convolutional layers, but it also contains other components: the poling layer, which is also known as downsampling, reduces the dimensions of each feature map and retains the most important information for the image. Similar to the convolutional layer, the pooling operation sweeps a filter across the entire input, but the difference is that this filter does not have any weights. Poling can be of different type: Max, Average,...etc. While the pooling layer loses a lot of information, it does have a few advantages for the CNN. They assist in reducing complexity, increasing efficiency and reducing the risk of overfitting.

The name of fully connected layers is self-explanatory, each node in the output layer is directly connected to a node in the previous layer, and this layer performs classification tasks (which are not the main task in our work), based on the features retrieved by the previous layers and their various filters (11).

Chapter 3

Deep neural network for solving inverse problem in imaging

In most applications, an observed signal y can be modeled as the output of a system T, whose input is denoted by x. as we have seen there are many ways to characterize the system T, through for instance, a differential equation, an integral equation, or a general mathematical mapping (12). It might model the image edge detection process or the defocusing introduced by the imaging device. Finding the input x for a given output y and knowledge of the system T represents the inverse problem and When the system is related to one of the tasks of image processing, here we are talking about inverse problem in imaging.

As shown in the figure 3.1 in the direct problem, the system T is applied to an input image. While the inverse problem aims to obtain an estimate of the input image, we show here the image restoration case where T represents the blurring operator.
In this chapter we discuss the problem where T is the noise factor known as the image denoising problem, and it is also a case of image restoration, we present a set of neural architecture to solve this problem, and we end with a practical section in which we discuss the application of one of these architectures to solve the problem at hand.

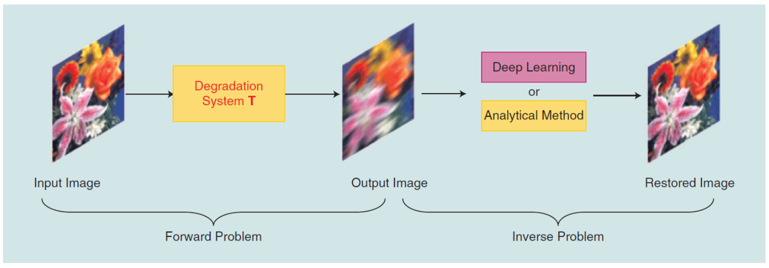


Figure 3.1: Moving from a direct problem to an inverse problem (13).

1. The concept of image processing

Image processing has become an exciting subject of study, due to the amazing diversity of applications it benefits from (figure 3.2).

In this section we will talk about a few fundamental definitions such as image, digital image, and digital image processing. we will talk also about the continuum from image processing to computer vision.

A digital image is a representation of a real image as a set of numbers that can be stored

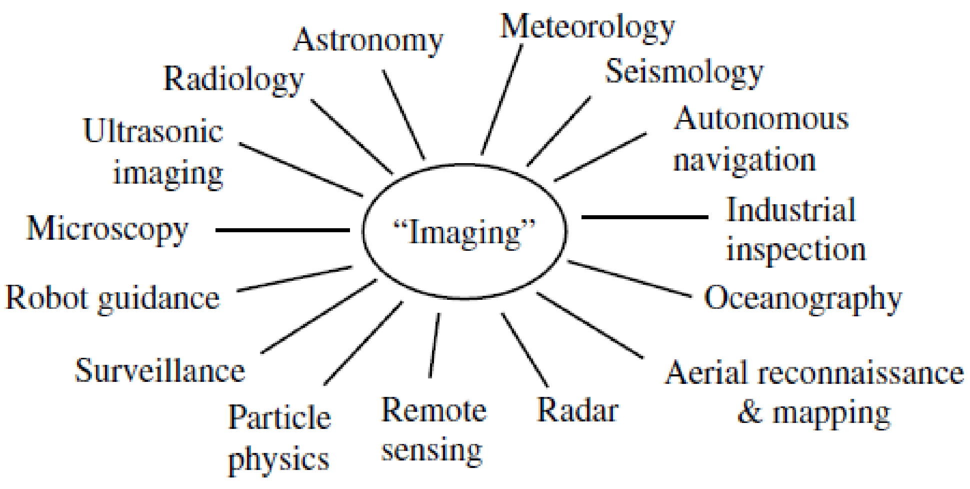


Figure 3.2: The amazing diversity of applications that benefit from image processing (14).

and handled In order to translate the image into numbers, it is divided into small areas

called pixels (picture elements). each pixel location only contains a single numerical value representing the signal level at that point in the image. For each pixel, the imaging device records a number, or a small set of numbers, that describe some property of this pixel, such as its brightness (the intensity of the light) or its color, an image contains one or more color channels that define the intensity or color at a particular pixel location. The numbers are arranged in an array of rows and columns that correspond to the vertical and horizontal positions of the pixels in the image.
Any meaningful two-dimensional matrix of integers can be considered an image from a mathematical point of view. In the real world, we need to be able to effectively display and store images for the purpose of using them in a range of tasks (15). the choice of image format used can be largely determined by not just the image contents, but also the actual image data type that is required for storage. there is a number of distinct image types:

1. **Grey-scale images** : are 2-D arrays that assign one numerical value to each pixel which is representative of the intensity at this point.
2. **Colour images (RGB)** : are 3-D arrays that assign three numerical values to each pixel, each value corresponding to the red, green and blue (RGB) image channel component respectively

As demonstrated in Figure 3.3, we can easily separate and view the red, green, and blue components of a true-color image. It's worth noting that the colors in a real image are almost always a combination of color components from all three channels.

An image can be regarded as a function f (x, y) of two continuous variables x and y, to be processed digitally, it has to be firstly sampled, sampling is the process of converting a continuous-space (or continuous-space/time) signal into a discrete-space (or discrete-space/time) signal. and transformed into a matrix of numbers. Secondly these numbers have to be quantized to be represented digitally, quantization is the process of converting

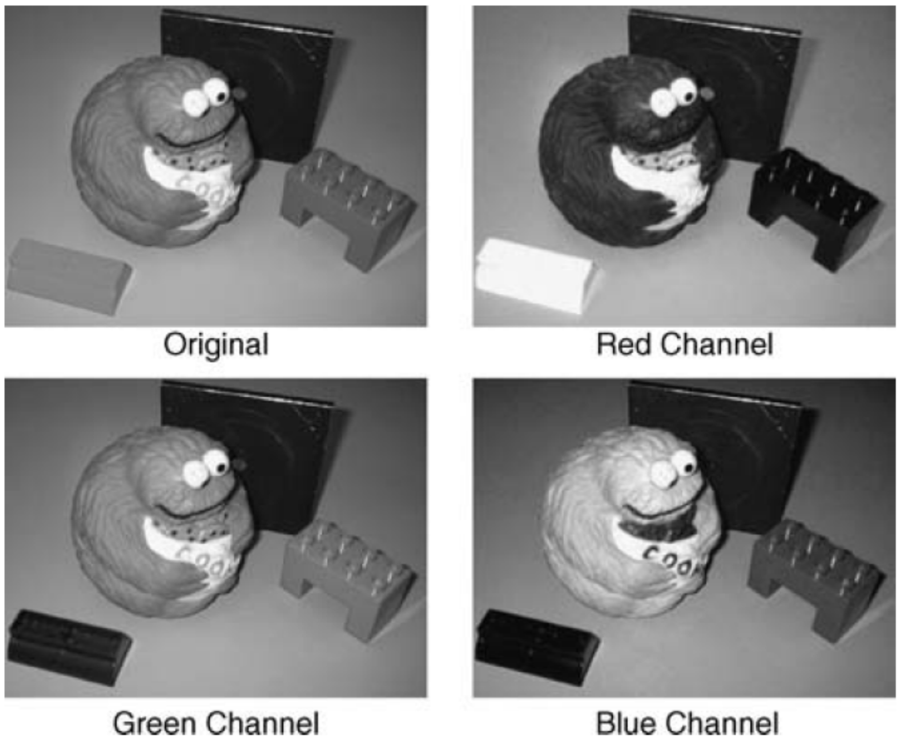


Figure 3.3: Separate and display the red, green and blue components of a true color image (15).

a continuous-valued image that has a continuous range (set of values that it can take) into a discrete-valued image that has a discrete range. This is ordinarily done by a process of rounding, truncation, or some other irreversible, nonlinear process of information destruction. that is necessary because a computer represents the numbers using finite precision, and the image intensities must be represented with a finite precision in any digital processor. The manipulation of those finite precision numbers is what digital image processing is all about.

One of the major issue in image processing, storage, transmission, and display is the large volume of image data, the amount of data in visual signals is typically fairly vast, and it grows geometrically in proportion to the data's dimensionality. The storage required for a single digital still image that has (row $\times$ column) dimensions and B bits of gray level resolution is (row × column × B) bits, digital images that are delivered by commercially available image digitizers are typically of approximate size 512×512 pixels. This what makes handling and analyzing a huge data set consisting of images so difficult.

Image enhancement, image restoration, image analysis, and image compression are some of the types of digital image processing (16). We shall concentrate on image restoration in our work as a must important application of inverse problem in imaging.

Image processing is related to many fields and has continuity and intersection with them on a number of levels. Computer vision is one of the most important fields that is considered an extension of image processing. Continuity from image processing to computer vision

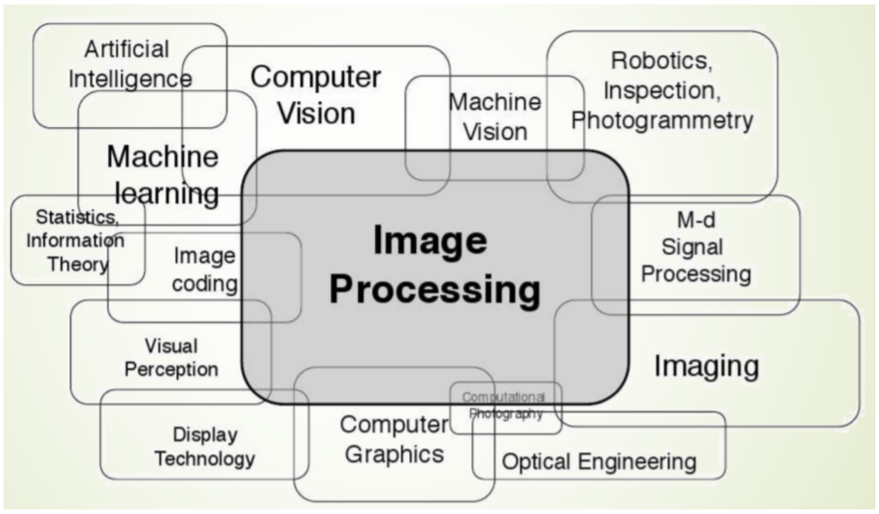


Figure 3.4: Image Processing Extensions (15).

can be divided into three levels of operations, the lowest level of which processes an image in the input and gives an image in the output, for example reducing noise on the image, while in the second level the image is processed in the input in order to obtain attributes for example object recognition, while at the third level based on the processing of attributes the system aim to create an understanding through them for example scene understanding and autonomous navigation (14).

In this work, we will address one of the applications of this continuity at its lowest level, since our system will be based on processing an image in the input and giving the clean image in the output.

2. Neural networks architectures for inverse problems in imaging

An alternative model for solving our inverse problems, considered as the most frequent approach, relies on the use of DNN for minimize $\|x - g_\phi(y)\|^2$ for a convenient $g_\phi(\cdot)$ which plays the role of T^{-1} we will show some designs for this function that correspond to a DNN with parameters $\mathbf{\Phi}$, and how to trained it using some data sets with pairs of examples (y, x).

The choice of the neural network architecture determines the generic set of alternative $g_\phi(\cdot)$ that will be investigated by the optimization technique for solving our inverse problems. as a result, it is critical to pay particular attention to the model's design, which will be the subject of our discussion in the following sections.

2.1. End-to-end mapping with the convolutional neural network

Due to DNNs' capacity to execute fast-forward inference, training DNNs to learn a mapping from the observation y to its reconstruction x is often the preferred strategy. a frequent architectural choice for performing this mapping was fully connected neural networks, such as the one shown in Figure.

The universal approximation theorem (17) states that a fully connected neural network with a large number of neurons in its hidden layer may represent any function we want to learn, as long as our activation functions meet some minor constraints. however when handling highly structured modalities like images or videos, a convolutional neural network (CNN) is often the preferred model.

CNNs are especially well-suited to image processing because they can quickly extract the

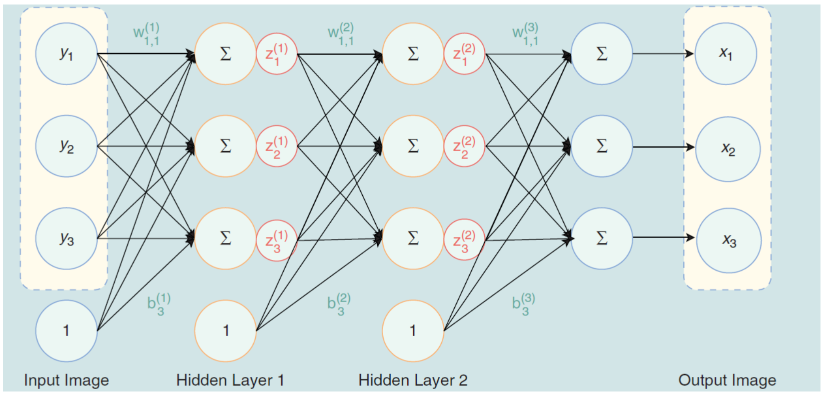


Figure 3.5: An example of a fully connected neural network with two hidden layers (13).

statistics of their input and utilize them to solve the inverse problem (18), the difference between it and fully connected neurons is that apply convolutions to the previous layer, on the input layer, several convolution kernels are used, resulting in various feature maps that collectively capture a new representation of the input.

Using CNNs to solve our inverse problems has a number of benefits. First, compared to fully connected neural networks, there are often far fewer parameters to learn because the weights of the kernels are fixed as they glide across the input. The optimization problem is made easier by the reduction in the number of parameters.

When we use neural networks to solve inverse imaging problems, the model produces a high-dimensional image with the same dimensions as the input.

As a result, keeping the dimensions of the output feature maps fixed to the size of the input to the convolutional layer, which is achieved by the use of proper padding with zeros, is a popular method when creating a CNN for addressing an inverse problem. the Figure shows an example of a three-layer CNN design that follows this technique.

CNN-based architectures, if properly designed, can be powerful tools for solving inverse problems in imaging.

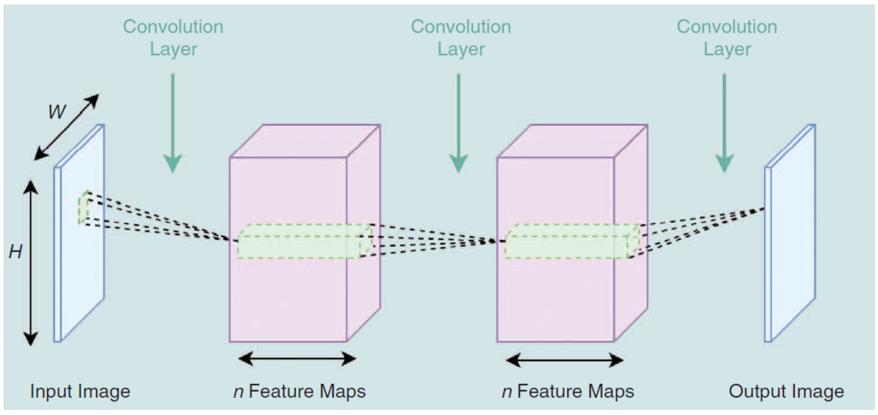


Figure 3.6: Convolutional layers, which maintain the same spatial dimensions of feature maps and those of input and output images (13).

2.2. Residual Networks (ResNet)

Different CNN architectures use more layers in a deep neural network to reduce the error rate. However as the number of layers increases, there is a typical problem in deep learning associated with the so-called Vanishing/Exploding gradient. When we increase the number of layers, the error rate of training and testing also increases.

In the graph below, we can see that a 56-layer CNN has a greater error rate on both training and testing dataset than a 20-layer CNN architecture, if this was the result of over fitting, then we should have lower training error in 56-layer CNN but then it also has higher training error. After analyzing more on error rate it was discovered that it is caused by vanishing/exploding gradient.

Resnet architecture proposed the concept of residual network to overcome this problem. Instead of learning a new mapping function from one layer to the next, residual blocks learn a residual between two or more layers by adding a skip connection from the input of the residual block to its output (19). It skips training from a few layers and connects directly to the output. the benefit of including this type of skip connection is that any layer that degrades architecture performance will be bypassed by regularization. As a result, very deep neural networks can be trained without the issues caused by vanishing/-

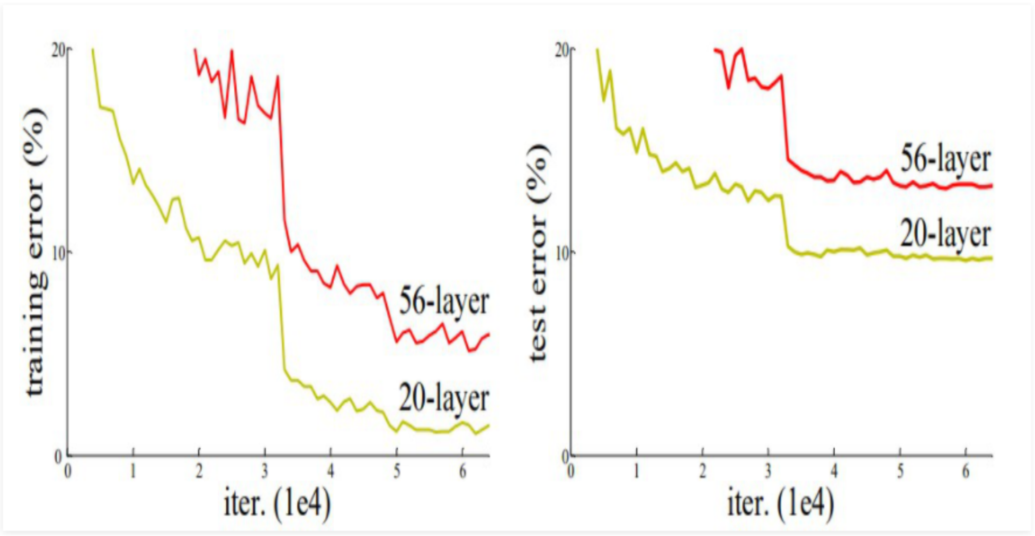


Figure 3.7: Vanishing gradient problem caused by increasing network depth.

exploding gradients.

A generic architecture of a deep residual CNN is shown in the Figure 3.8 to explain this concept. Where each residual block, consisting here of three convolutions, learns a residual between its input and its output.

Deep residual networks can be regarded of as a stable alternative to training DNNs

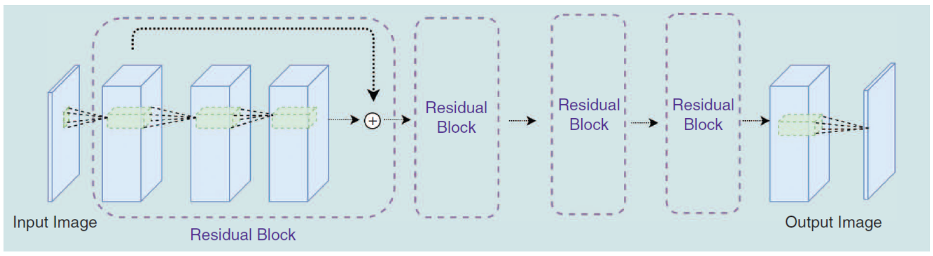


Figure 3.8: An example of a deep residual CNN. Each residual block, consisting here of three convolutions, learns a residual between its input and its output (13).

because learning residual is easier than learning a new mapping from layer to layer (20).

2.3. Encoder-decoder architecture

The more we discover about deep learning's success, the more enigmatic deep neural networks become. in particular in inverse problems, the so-called encoder-decoder CNN is one of the most extensively used network architectures, these encoder-decoder CNNs frequently have more intricate network designs, such as symmetric network setup, skipped connections, and so on, in contrast to the basic form of neural networks that is often employed (21).

Encoder-decoder CNNs distinguish themselves from the networks previously mentioned by choose to downsample the feature, maps at each convolution step all the way down to a bottleneck layer, and then upsample them back to the size of the output, as shown in the Figure 3.9. While the networks previously mentioned keep the dimensions of the feature maps fixed to the dimension of the input and output images.

We can see in the architecture that the feature maps are spatially compressed by an encoder network (Pooling) , then increased back to the size of the output image by a decoder network(Upsamplig) . Downsampling and upsampling feature maps has gained popularity in segmentation and depth prediction applications (the architecture illustrated in the figure is for a segmentation task) .The "compressive" portion of the network learns an abstract representation of the input image, which is then used by the "expansive" portion of the network to generate an output image.

An encoding-decoding architecture may result in a significant loss of detail in the output image because the encoder compresses the spatial information of the feature maps at each step. Inserting symmetric skip connections between the lower-downsampling convolutional layers of the network and the matching up-sampling convolutional layer, which preserves the crucial features in the input image, is one solution to this problem. We should note that the encoder-decoder architecture with skip connections is commonly referred to as the U-Net architecture (22), which we will discuss in greater depth in the following section because it is the architecture we will utilize in our application.

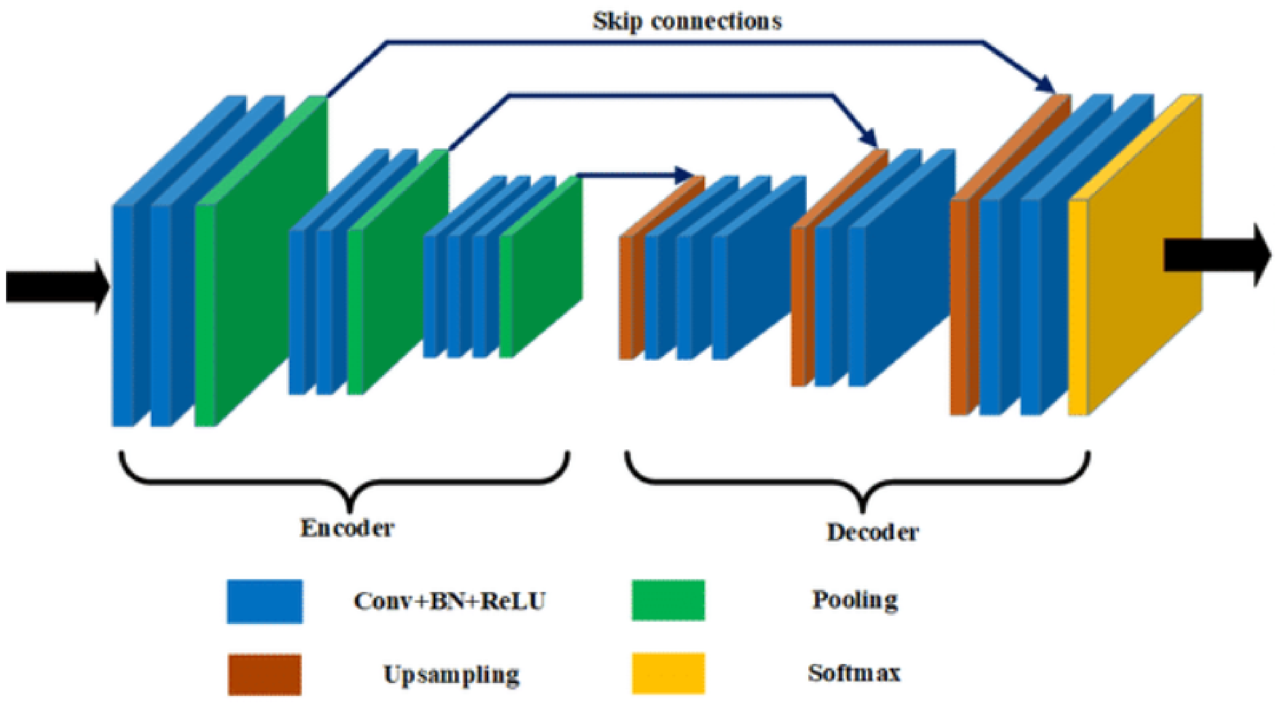


Figure 3.9: The encoder (Pooling) and decoder network (Upsampling) components (22).

This encoder-decoder CNN or U-Net has been effectively used to a variety of inverse imaging problems, including the image denoising problem which we will go over in detail.

3. Image denoising with deep CNN

3.1. Introduction

There are various image denoising models that have recently attracted great interest due to their positive noise reduction performance (23). In this section, we try to use deep convolutional neural networks (CNNs) with PyTorch in the form of U-Net architectures to solve an image denoising problem (Gaussian denoising). Specifically, residual learning and batch normalization are utilized to speed up the training process as well as improve the denoising performance.

In the first step, we will deepen our knowledge of the deep learning architecture that we will work with (U-Net), and one of the main software libraries for CNN's application of image processing called PyTorch. In the second step, we use an already available data set (BSDS300) to train and test our U-Net model and analyze the results received.

Image restoration problem:

Inverse problems arise in a variety of imaging applications such as inpainting and super resolution (SR), one of the must important application is image reconstruction which aimed at restoring the original signal, is the process of restoring original uncorrupted images from corrupted ones. Motion blur, low resolution, and noise are all examples of corruption. The mean goal of image reconstruction is to improve the quality of an image by understanding the physical processes that lead to its creation (24). Image formation can be thought of as a process that converts an input distribution into an output distribution. The input distribution represents the ideal or perfect image to which we do not have direct access, but seek to restore or at least approximate by treating the imperfect or corrupted output distribution appropriately. The task of recovering the image (in principle at least) is a simple equation that we can model as :

$$g(x,y) = \iint f\left(x', y'\right) h\left(x - x', y - y'\right) \mathrm{d}x' \mathrm{d}y' + n(x,y) \tag{3.1}$$

Estimate the input distribution $f(x', y')$ using the measured output $g(x,y)$ and any knowledge we may possess about the PSF $h(x - x', y - y')$ and the noise term $n(x,y)$. These two factors are responsible for the imperfect output distribution which is obtained. We have to give an overview to both since the simple remark that any strategy to image reconstruction, will ultimately be constrained by our explicit or tacit understanding of PSF and the noise process.

Nature of the point-spread function and noise

The PSF is typically a constant or deterministic number that is determined by the physical hardware which constitutes the overall imaging system and which may, therefore, be considered to be stable with time. For example, in a simple optical imaging system, the overall PSF will be defined by the physical nature and shape of the lenses; in a medical diagnostic imaging system, such as the Anger camera (which detects gamma-rays emitted from the body of a patient), the PSF is determined by a combination of a mechanical collimator and a scintillator– photomultiplier system which allows the detection of the origin of an emitted gamma photon with a certain limited accuracy.

The noise term on the other hand, is often stochastic and causes a random and undesirable fluctuation in the measured signal. The main feature of noise processes is that we usually have no influence over them and can't foresee what kind of noise will be present at any particular time. Noise is created by the physical nature of detection processes and comes in a variety of shapes and sizes. The unpredictable nature of the signal fluctuations is a common aspect of whatever physical process leads to the development of an image. we can often understand and model the statistical features of a given realization of the noise, even if we can't know the values of that realization. The various noise models and behaviors are frequently a major element in the subtle differences in image restoration procedures.

Denoising problem

Denoising image is a technique for recovering a high-quality image from a noisy (degraded) observation. It's one of the most well-known and essential image processing and computer vision problems, where user experience and the success of high-level vision tasks such as object identification and recognition are affected by the quality of the resulting images. A simplified general image degradation model for the denoising task is:

$$y = x + n \tag{3.2}$$

where x refers to the unknown high-quality image or clean image, y is noisy observation, and n represents the additive noise. For decades, most of the noise reduction research has been done on the additive white gaussian noise (AWGN) box it is Probably the most frequently occurring noise, which we will also rely on in this work.

AWGN assumes that n is the symmetrically distributed Gaussian noise with zero mean and standard variance value . The Gaussian distribution has a number of useful mathematical features as well as a few less useful ones. The fact that the cumulative distribution function cannot be written in closed form using elementary functions is undoubtedly the least convenient property of the Gaussian distribution. It is however numerically tabulated, perhaps the most significant property of the Gaussian distribution is called the Central Limit Theorem, which states that the distribution of a sum of a large number of independent, small random variables has a Gaussian distribution.

The major problem with image denoising is that a large amount of information is lost

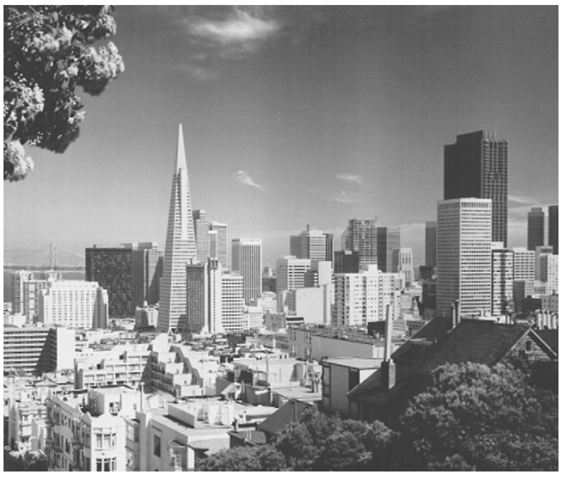


Figure 3.10: Original picture of San Francisco skyline (15).

throughout the degrading process, making it an inverse problem with a high degree of ill-posedness. Prior knowledge is necessary to offer supplemental information in order to obtain an accurate estimation of the latent image. as a result, therefore how to appropriately model the prior of high quality (clean) images is the main difficulty in image

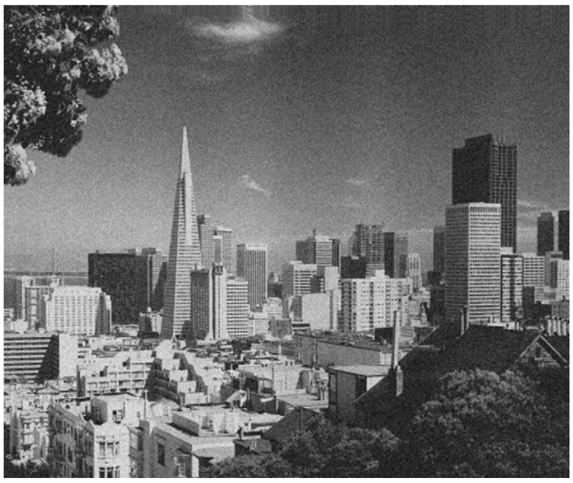


Figure 3.11: San Francisco corrupted by additive Gaussian noise with $\sigma = 10$ (15).

reconstruction problems.

3.2. works tools

In this section we introduce PyTorch as a library and its place in the deep learning revolution, touching on what distinguishes PyTorch from other deep learning frameworks, and we describe the network architecture of the model used for this application.

PyTorch:

PyTorch is an open source machine learning framework, and an optimized tensor library primarily used for deep learning applications using GPUs and CPUs. Mainly developed by the Facebook AI research team.

PyTorch is software for machine learning it contains a full tool kit for building and deploying machine learning applications including deep learning primitives (25), it comes with features to perform accelerated mathematical operations on dedicated hardware, which makes it convenient to design neural network architectures and train them on individual machines or parallel computing resources.

The concept of tensors is the center of everything doing in Pytorch like model's inputs outputs, and learning weights are all in form of tensors when we say tensors we talking about a multi-dimensional array with a lot of extra bells and whistles. It shares many similarities with NumPy arrays. PyTorch has a reputation for simplicity, ease of use, flexibility, efficient memory usage and dynamic computational graphs. It also feels native, making coding more manageable and increasing processing speed. It is one of the widely used machine learning libraries, others being TensorFlow and Keras.

PyTorch is built based on python and torch library, Torch.nn contains the main PyTorch modules for building neural networks, as well as common neural network layers and other architectural components, such as fully connected layers, convolutional layers, activation functions and loss functions. These components can be used to create and initialize the untrained model in the figure.

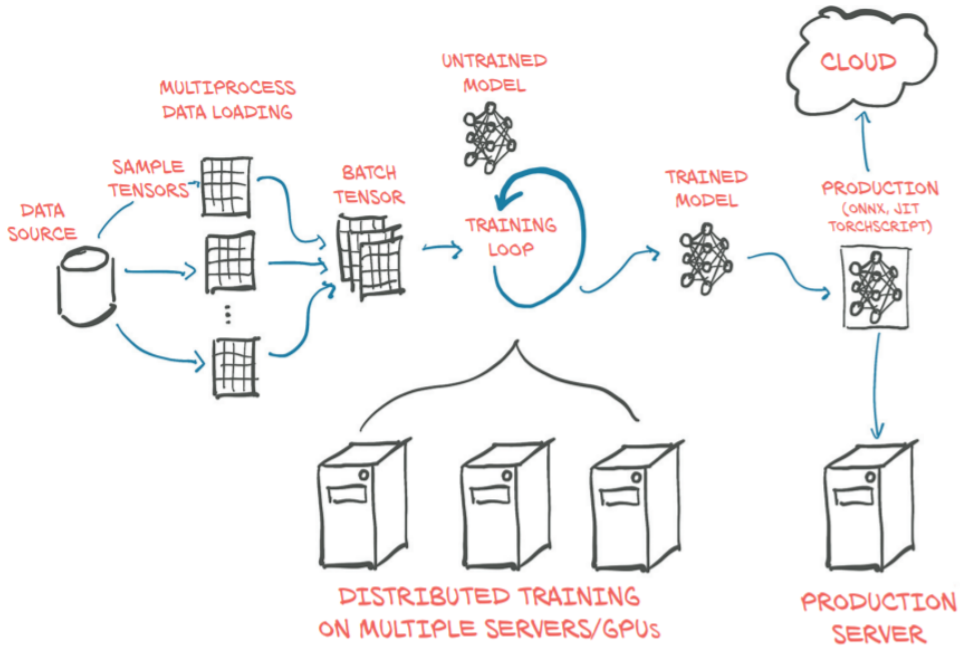
To train our model, we'll need a source of training data, an optimizer to adjust the model to the training data and a way to send the model and data to the hardware that will actually be performing the calculations..

Network architectute of the used denoising model

In this section, we present the used denoising model UDnCNN (26), which is a modification of the DnCNN model (27). It is the first model founding that the residual learning integration and batch normalization can lead to fast and stable training and better denoising performance.

The input of UDnCNN is a noisy observation, the model aim to learn a mapping function $F(y) = x$ with depth D to predict the latent clean image in the output, there are three types of layers, shown in the Figure.

1. **Conv+ReLU** : for the first layer, 64 filters of size 3 × 3 × c are used to generate 64 feature maps, and rectified linear units (ReLU, $max(0, ů)$) are then utilized for nonlinearity. Here c represents the number of image channels, (c = 1 for gray image and c = 3 for color image)



MULTIPROCESS DATA LOADING
UNTRAINED MODEL
CLOUD
SAMPLE TENSORS
DATA SOURCE
BATCH TENSOR
TRAINING LOOP
TRAINED MODEL
PRODUCTION (ONNX, JIT TORCHSCRIPT)
DISTRIBUTED TRAINING ON MULTIPLE SERVERS/GPUs
PRODUCTION SERVER

Figure 3.12: Basic, high-level structure of a PyTorch project, with data loading, training (25).

2. **Conv+BN+ReLU+Pool** : the layers 2 ~ $(D/2)$, represent The "compressive" portion of the network(Encoder) that learns an abstract representation of the input image using the Poolin layes that is much like a filter applied to feature maps, it is 2×2 pixels applied with a stride of 2 pixels. This means that the pooling layer will always reduce the size of each feature map by a factor of 2, each dimension is halved. and batch normalization (27) is added between convolution and ReLU.

3. **Unpool+ Conv+BN+ReLU** : represent the "expansive" portion of the network(decoder) which use the output of the compressive portion previously mentioned and seek to restore the lost features and restore the original dimension of the input image by adding an Unpooling layer to generate an output image.

4. **Conv** : the last layer is a simple convolution layer that use , c filters of size 3 × 3 × 64 are to reconstruct the output.

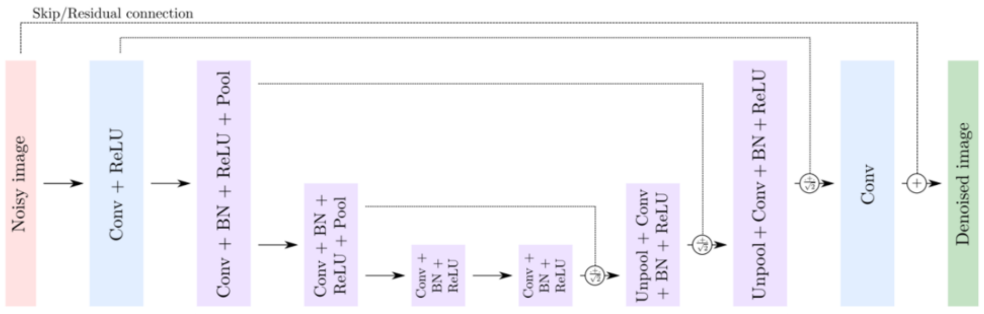


Figure 3.13: UDnCNN Model architecture (26).

In the architecture we can see the insertion of residual connection and batch normalization (28) for speed up and stabilize the training process and also for boosting the denoising performance.

3.3. Experimental Results

Our experiments were based on three main stages:

1. we prepared the dataset BSDS300 (27) It consists of 200 images for training and 100 images for testing, by adding Gaussian noise to the clean images In the training and test set, we consider three noise levels: $\sigma = 15$(low) $\sigma = 25$(medium) and $\sigma = 50$(high)
2. we run the experiment by training the network with backpropagation based on the optimizer and training set.
3. we evaluated the experiment by forwarding the test set over the network and returning the computed statistics

With the ultimate goal of reducing image noise, we need to evaluate the quality of the image obtained for this purpose we used pixel-wise Mean Squared Error.

$$L(d, y) = \frac{1}{N} \sum_{i=0}^{N} (d - y_i)^2 \tag{3.3}$$

For each image in training set d, we apply a set Gaussian noise to get noisy images x. The resulting image is entered into the proposed model for training; then the resulting

Table 3.1: The average PSNR(dB) and loss function results of DnCNN/UDnCNN models with noise level 15, 25 and 50 on BSDS300 dataset

	DnCNN		UDnCNN(D=6)		UDnCNN(D=8)	
Noise level	PNSR	SSIM	PNSR	loss	PNSR	loss
15	31.46	0.8826	32.04	0.0025	32.03	0.0026
25	29.02	0.8190	29.13	0.0049	29.13	0.0049
50	26.10	0.7076	25.30	0.011	25.21	0.012

image y is compared with the clean image d by means of the MSE (the lower the better).

An other classical way to compare the quality of restoration techniques that we used is the PSNR (Peak Signal-to-Noise-Ratio) defined as :

$$PSNR = 10 \log_{10} \frac{4n}{\|y - d\|_2^2} \tag{3.4}$$

where d is the desired ideal image, y is the estimate obtained from x and n the number of elements in the tensor. since many signals have a very wide dynamic range, PSNR is it is usually expressed in terms of the logarithmic scale of decibels (dB). the higher the PSNR value, the higher similarity to the original image.

The table 3.1 displays the results obtained through training and testing the given architecture with two different depths and a variation in thetab1 added noise. We followed the steps in this tutorial (26). and worked on changing some parameters and monitoring the results and observations. We did almost the same experiments and worked on analyzing the results.

We note through the observed results in the table that the U-net architecture helps greatly in giving greater efficiency in denoising problems, but the problem of determining the depth remains. Through the comparison we made, it was found that the appropriate depth for the model is D=6 and that the increase in noise significantly affects its performance.

```
{'loss': 0.004913777858018875, 'PSNR': tensor(29.1387)}
```

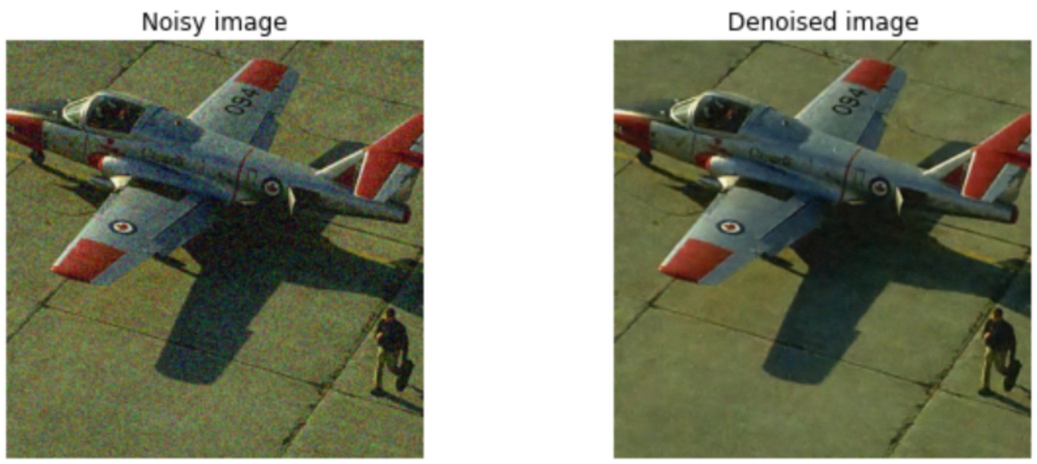


Figure 3.14: The output of the UDnCNN model in the training phase with D = 6 and, σ = 25.

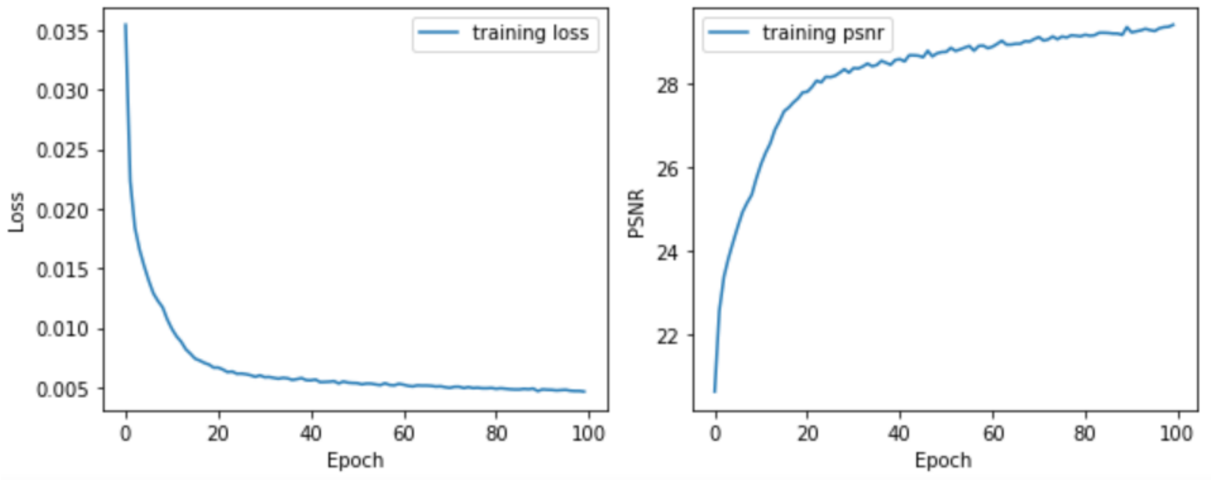


Figure 3.15: Change of PNSR and loss function based on the change of epoch during the training phase.

The two figures 3.14 and 3.15 represent the results obtained using the model by adding a noise level σ = 25. We chose 100 as a maximum of number of epoch because, as we note in the figure, after this number, there is no significant change at the level of the two index PNSR and loss function.

Conclusions

The present thesis aims to study the inverse problems by highlighting the diversity in the ways to solve them, as we presented in the first part some classic methods through a theoretical study in cooperation with Professor G.Rodriguez, via a comprehensive review of the specific literature and specialized material presented in the references.

In the first chapter we introduced various methods of regularization for calculating approximate solutions.we have covered three regularization approaches , but we have discovered that the most important thing is to find an effective method for determining the regularization parameter. Because it is the only way to set up an effective regularization method. which lead to a reasonable balance of perturbation and regularization errors in the solution.

While in the second part we have highlighted other Data-driven approaches that deal with inverse problems. this was a result of an internship in collaboration with Professor G.Fumera in the Department of Electronic and Electrical Engineering at the Faculty of Engineering in Cagliari. we used the deep learning methods and architectures to solve inverse problems in the image reconstruction field.

The first stage, is about extending the knowledge in the deep learning architectures as one of the most important modern tools for solving inverse, that we presented in the second chapter by highlighting the fundamental of deep learning and how to train a deep neural networks, especially Convolutional Neural Networks (CNNs), and the study of one of the main software libraries of CNN application for image processing (PyTorch and TensorFlow).

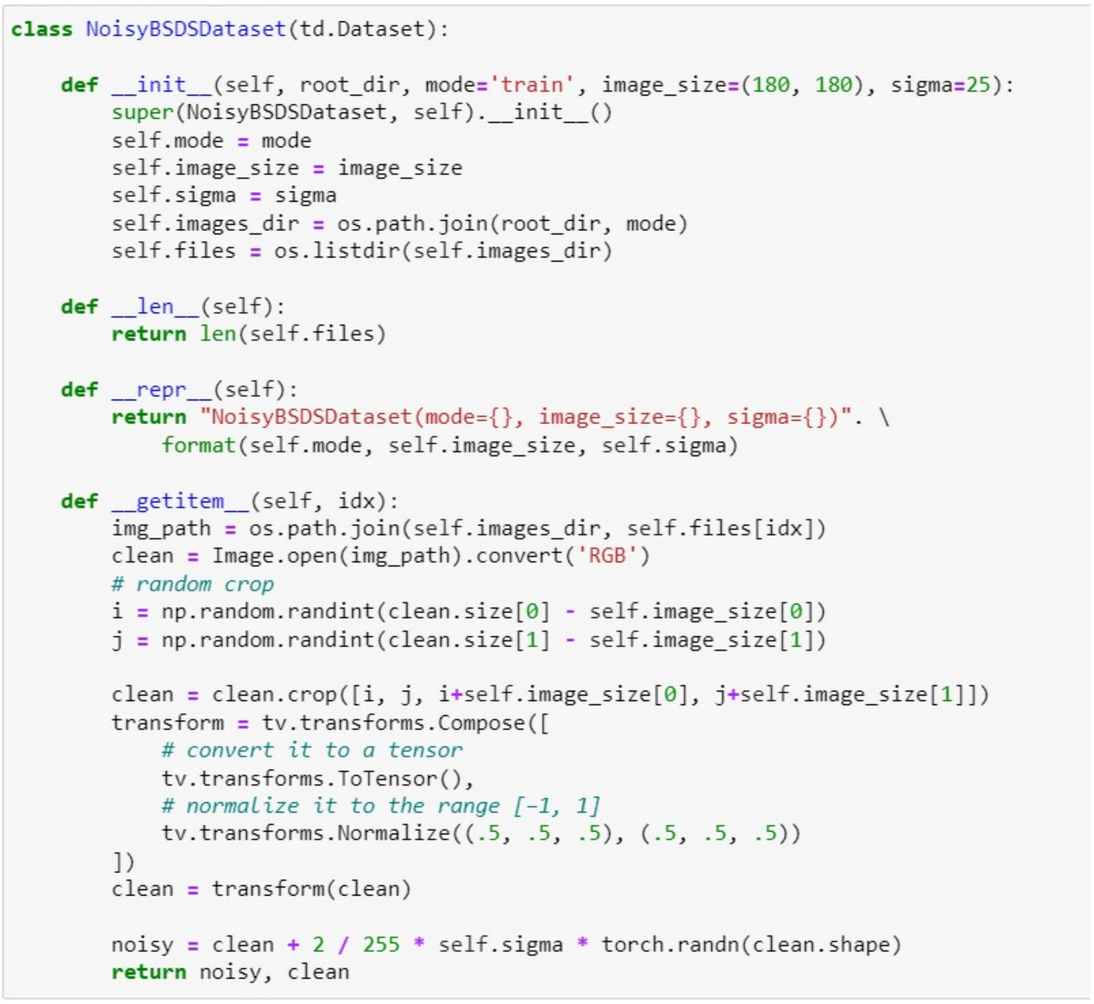
The second stage, is more about identifying the problem of image denoising as one of the most famous inverse problems in imaging as a practical application, which we studied in the therd chapter.

While studying U-Net as one of the effective architecture for image processing, especially the task of image segmentation , we discovered through the application that it gives a good performance even in the task of removing noise from the image by comparing it with the classical CNN models, yet there are some challenges represented in determining the parameters related to the model architectures such as depth and dimensions.

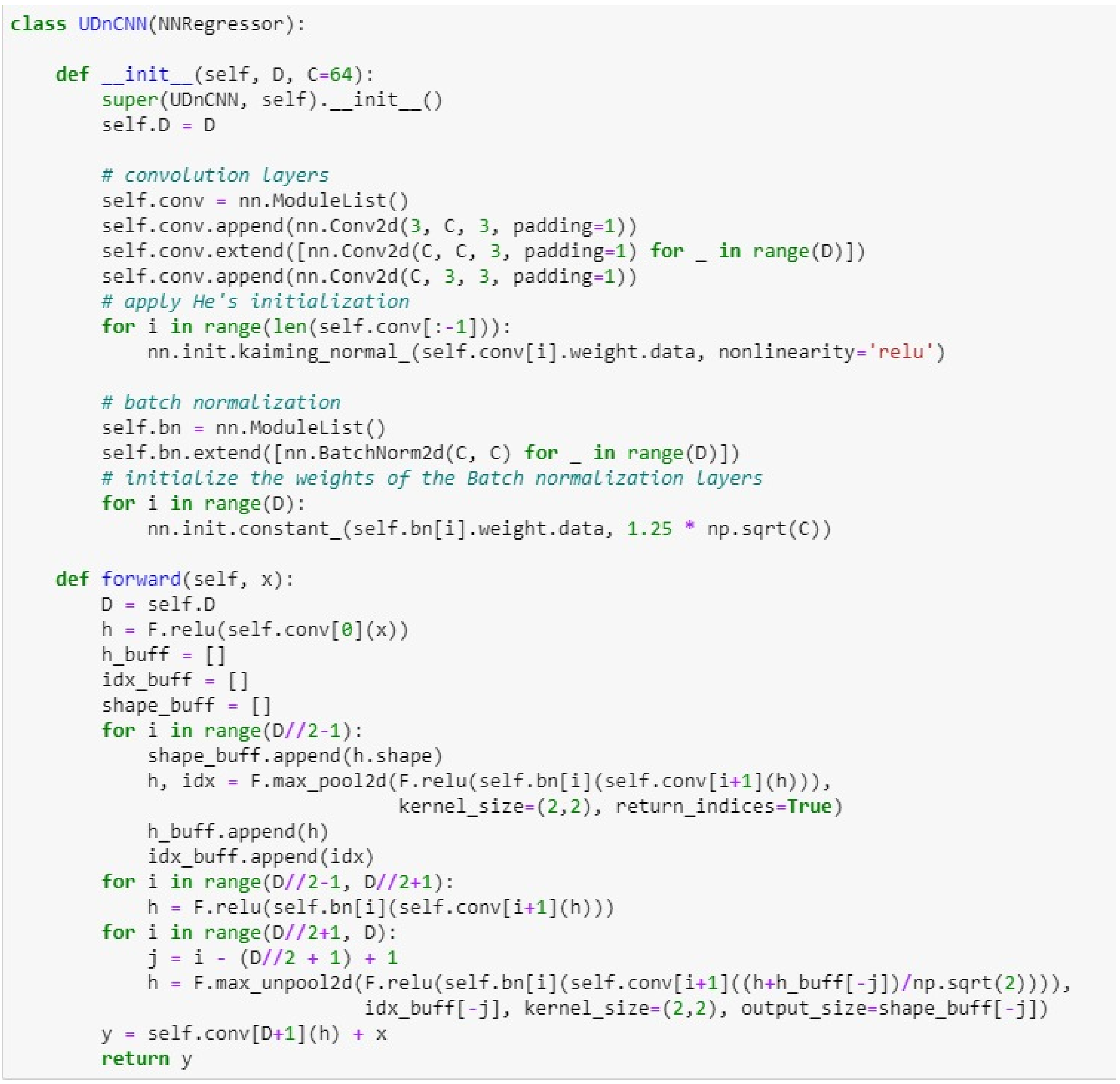
Appendix A

Appendix

To give an insight into the technical work performed to get all the resulted tables and processed images presented in chapter 3, the following appendix shows through code and plots, the application of UDnCNN model to solve the image denoising problem.



```python
class NoisyBSDSDataset(td.Dataset):

    def __init__(self, root_dir, mode='train', image_size=(180, 180), sigma=25):
        super(NoisyBSDSDataset, self).__init__()
        self.mode = mode
        self.image_size = image_size
        self.sigma = sigma
        self.images_dir = os.path.join(root_dir, mode)
        self.files = os.listdir(self.images_dir)

    def __len__(self):
        return len(self.files)

    def __repr__(self):
        return "NoisyBSDSDataset(mode={}, image_size={}, sigma={})". \
            format(self.mode, self.image_size, self.sigma)

    def __getitem__(self, idx):
        img_path = os.path.join(self.images_dir, self.files[idx])
        clean = Image.open(img_path).convert('RGB')
        # random crop
        i = np.random.randint(clean.size[0] - self.image_size[0])
        j = np.random.randint(clean.size[1] - self.image_size[1])

        clean = clean.crop([i, j, i+self.image_size[0], j+self.image_size[1]])
        transform = tv.transforms.Compose([
            # convert it to a tensor
            tv.transforms.ToTensor(),
            # normalize it to the range [-1, 1]
            tv.transforms.Normalize((.5, .5, .5), (.5, .5, .5))
        ])
        clean = transform(clean)

        noisy = clean + 2 / 255 * self.sigma * torch.randn(clean.shape)
        return noisy, clean
```

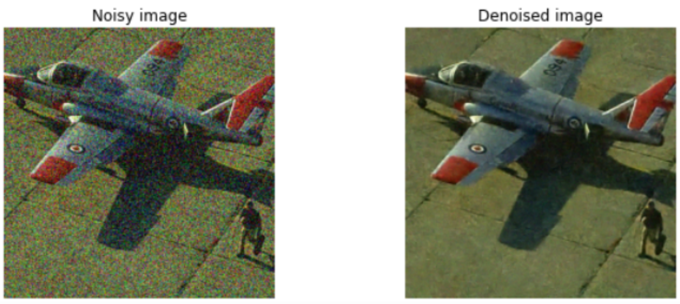


```python
class UDnCNN(NNRegressor):

    def __init__(self, D, C=64):
        super(UDnCNN, self).__init__()
        self.D = D

        # convolution layers
        self.conv = nn.ModuleList()
        self.conv.append(nn.Conv2d(3, C, 3, padding=1))
        self.conv.extend([nn.Conv2d(C, C, 3, padding=1) for _ in range(D)])
        self.conv.append(nn.Conv2d(C, 3, 3, padding=1))
        # apply He's initialization
        for i in range(len(self.conv[:-1])):
            nn.init.kaiming_normal_(self.conv[i].weight.data, nonlinearity='relu')

        # batch normalization
        self.bn = nn.ModuleList()
        self.bn.extend([nn.BatchNorm2d(C, C) for _ in range(D)])
        # initialize the weights of the Batch normalization layers
        for i in range(D):
            nn.init.constant_(self.bn[i].weight.data, 1.25 * np.sqrt(C))

    def forward(self, x):
        D = self.D
        h = F.relu(self.conv[0](x))
        h_buff = []
        idx_buff = []
        shape_buff = []
        for i in range(D//2-1):
            shape_buff.append(h.shape)
            h, idx = F.max_pool2d(F.relu(self.bn[i](self.conv[i+1](h))),
                                  kernel_size=(2,2), return_indices=True)
            h_buff.append(h)
            idx_buff.append(idx)
        for i in range(D//2-1, D//2+1):
            h = F.relu(self.bn[i](self.conv[i+1](h)))
        for i in range(D//2+1, D):
            j = i - (D//2 + 1) + 1
            h = F.max_unpool2d(F.relu(self.bn[i](self.conv[i+1]((h+h_buff[-j])/np.sqrt(2)))),
                               idx_buff[-j], kernel_size=(2,2), output_size=shape_buff[-j])
        y = self.conv[D+1](h) + x
        return y
```

```
D=6

device = 'cuda' if torch.cuda.is_available() else 'cpu'
lr = 1e-3
net = UDnCNN(6).to(device)
adam = torch.optim.Adam(net.parameters(), lr=lr)
stats_manager = DenoisingStatsManager()
exp = Experiment(net, train_set, test_set, adam, stats_manager, batch_size=4, perform_validation_during_training=True)
```

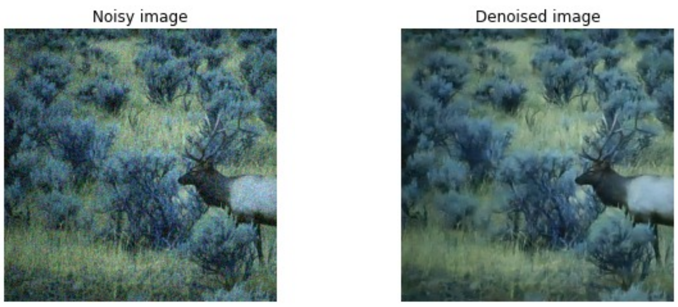
```
fig, axes = plt.subplots(ncols=2, nrows=2, figsize=(9, 7))
exp.run(num_epochs=100, plot=lambda exp: plot(exp, fig=fig, axes=axes,
                                              noisy=test_set[70][0]))
exp.evaluate()
```

```
Epoch 99 | Time: 92.99s | Training Loss: 0.011236 | Evaluation Loss: 0.012161
Epoch 100 | Time: 93.70s | Training Loss: 0.011368 | Evaluation Loss: 0.011943
Finish training for 100 epochs

{'loss': 0.011898165158927441, 'PSNR': tensor(25.3048)}
```



```
#for D=8

device = 'cuda' if torch.cuda.is_available() else 'cpu'
lr = 1e-3
net = UDnCNN(8).to(device)
adam = torch.optim.Adam(net.parameters(), lr=lr)
stats_manager = DenoisingStatsManager()
exp = Experiment(net, train_set, test_set, adam, stats_manager, batch_size=4, perform_validation_during_training=True)

fig, axes = plt.subplots(ncols=2, nrows=2, figsize=(9, 7))
exp.run(num_epochs=100, plot=lambda exp: plot(exp, fig=fig, axes=axes,
                                              noisy=test_set[72][0]))
exp.evaluate()
```

```
[illegible]
Epoch 100 | Time: 167.24s | Training Loss: 0.004768 | Evaluation Loss: 0.004938
Finish training for 100 epochs

{'loss': 0.004928439417853952, 'PSNR': tensor(29.1354)}
```



Bibliography

[1] P. C. Hansen, *Discrete inverse problems: insight and algorithms.* SIAM, 2010.

[2] G. Wang, J. C. Ye, K. Mueller, and J. A. Fessler, "Image reconstruction is a new frontier of machine learning," *IEEE transactions on medical imaging*, vol. 37, no. 6, pp. 1289–1296, 2018.

[3] S. Arridge, P. Maass, O. Öktem, and C.-B. Schönlieb, "Solving inverse problems using data-driven models," *Acta Numerica*, vol. 28, pp. 1–174, 2019.

[4] M. Kern, *Numerical methods for inverse problems.* John Wiley & Sons, 2016.

[5] L. Rosasco, A. Caponnetto, E. Vito, F. Odone, and U. Giovannini, "Learning, regularization and ill-posed inverse problems," *Advances in Neural Information Processing Systems*, vol. 17, pp. 1145–1152, 2004.

[6] F. Chollet, *Deep learning with Python.* Simon and Schuster, 2021.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016, http://www.deeplearningbook.org.

[8] M. A. Nielsen, *Neural networks and deep learning.* Determination press San Francisco, CA, 2015, vol. 25.

[9] I. C. Education, "Convolutional neural networks," https://www.ibm.com/cloud/learn/convolutional-neural-networks, 2021.

[10] S. R. Arridge, V. Maarten, P. Maaß, and C.-B. Schönlieb, "Mini-workshop: Deep

learning and inverse problems," *Oberwolfach Reports*, vol. 15, no. 1, pp. 559–589, 2019.

[11] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," *arXiv preprint arXiv:1412.7062*, 2014.

[12] G. Ongie, A. Jalal, C. A. Metzler, R. G. Baraniuk, A. G. Dimakis, and R. Willett, "Deep learning techniques for inverse problems in imaging," *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 39–56, 2020.

[13] A. Lucas, M. Iliadis, R. Molina, and A. K. Katsaggelos, "Using deep neural networks for inverse problems in imaging: beyond analytical methods," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 20–36, 2018.

[14] A. C. Bovik, *The essential guide to image processing.* Academic Press, 2009.

[15] C. Solomon and T. Breckon, *Fundamentals of Digital Image Processing: A practical approach with examples in Matlab.* John Wiley & Sons, 2011.

[16] M. Bertero and P. Boccacci, *Introduction to inverse problems in imaging.* CRC press, 2020.

[17] K. Hornik, M. Stinchcombe, and H. White, "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks," *Neural networks*, vol. 3, no. 5, pp. 551–560, 1990.

[18] K. H. Jin, M. T. McCann, E. Froustey, and M. Unser, "Deep convolutional neural network for inverse problems in imaging," *IEEE Transactions on Image Processing*, vol. 26, no. 9, pp. 4509–4522, 2017.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[20] X. Mao, C. Shen, and Y.-B. Yang, “Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections,” *Advances in neural information processing systems*, vol. 29, pp. 2802–2810, 2016.

[21] H. Latha and R. R. Sahay, “A local modified u-net architecture for image denoising,” *reconstruction*, vol. 8, p. 14, 2020.

[22] M. Livne, J. Rieger, O. U. Aydin, A. A. Taha, E. M. Akay, T. Kossen, J. Sobesky, J. D. Kelleher, K. Hildebrand, D. Frey *et al.*, “A u-net deep learning framework for high performance vessel segmentation in patients with cerebrovascular disease,” *Frontiers in neuroscience*, vol. 13, p. 97, 2019.

[23] C. Tian, L. Fei, W. Zheng, Y. Xu, W. Zuo, and C.-W. Lin, “Deep learning on image denoising: An overview,” *Neural Networks*, 2020.

[24] H. Li, J. Schwab, S. Antholzer, and M. Haltmeier, “Nett: Solving inverse problems with deep neural networks,” *Inverse Problems*, vol. 36, no. 6, p. 065005, 2020.

[25] E. Stevens, L. Antiga, and T. Viehmann, *Deep learning with PyTorch*. Manning Publications, 2020.

[26] L.-Y. Cheng, “Image-denoising-with-deep-cnns,” https://github.com/lychengr3x/Image-Denoising-with-Deep-CNNs, Jul. 2020.

[27] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, “Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising,” *IEEE transactions on image processing*, vol. 26, no. 7, pp. 3142–3155, 2017.

[28] R. Komatsu and T. Gonsalves, “Comparing u-net based models for denoising color images,” *AI*, vol. 1, no. 4, pp. 465–486, 2020.