MASTER'S THESIS

Control system for mirror tilting by deep learning

Department of Physics, School of Science,

Tokyo Institute of Technology

Yilun Hou

July 11, 2023

Abstract

In gravitational wave detection, complex optical systems are built to reach the precision of a gravitational wave signal. There are multiple noises that affect the precision of the detector. When a laser hits a mirror or other components of an optical system, it will cause a movement or rotation to that component. The traditional way of solving this problem can be divided into two steps. First, when a PDH signal is detected by a single detector, we can lock the beam and stabilize it. Then a detection on WFS signals is possible and certain Hermite Gaussian modes can be separated out of the beam. Usually, the 00 part and 10 part of Hermite Gaussian modes are taken and used for controlling the mirror.

Previous study shows that deep learning is useful in developing such an automatic control system with traditional calculation. A convolutional neural network (CNN) has been built to analyse WFS signal from a Fabry-Perot cavity. The WFS signal data set is first pre-processed by principal component analysis (PCA), where those signals are reorganized into sets of eigenvalues of a few new bases. The data set is then put into the CNN model and trained with 5000 epochs. Result shows the loss of that deep learning model can be reduced to a reasonable range.

In this study, we take one step further to this simulation of Fabry-Perot cavity. We first introduced the roughness of mirrors into the optical system. Beside CNN model, we also added the linear artificial neural network (ANN) into comparison. The result is interesting. We built two different deep learning models that have an accuracy over 90% against rotations around 1 micron radian. This precision may not be enough for the current situation in gravitational wave detection, but we believe there are lots of ways to further optimize the current models.

Contents

Chapter 1

Introduction

Since Einstein opened the first page of relativity, human's vision is now one step further and deeper into the universe. Just like the lights bring us the information from the sky, gravitational waves also bring us the information from the universe, more but weaker. Since then, large and complex optical system are built for gravitational wave detection.

Based on the theory of Michelson interferometer, ground-based gravitational wave detectors have been built to detect gravitational waves from the universe. To finer the precision of the detection, an array of this kind of detectors has been built. It is consist of Advanced LIGO [1] in Hanford and Livingston in United States, Virgo [2] in Europe, and KAGRA [3] in Japan. For such a precise optical system, there are different types of noise that affect the information from gravitational waves. Shot noise, radiation pressure noise, thermal noise and seismic noise. When we focus on laser itself, we can see the radiation pressure noise occurs and keep pressing and causes movement and tilting to the laser. Currently, there are multiple ways to calculate and reduce the effect of radiation pressure noise. Such as traditional alignment or calculating Wave-front sensing (WFS) signal with Pound–Drever–Hall (PDH) locking technique. However, both method take a long time to react. Therefore, the aim of this study is to build an automatic control system that allows the optical system adjusting itself in real time. In optical systems, we usually use mirrors to reflect and transmit the laser and pass the laser to other optical components.

In gravitational wave detection, There are multiple noises that affect the result from a detector. Seismic noise is one of these noises and it is caused by the surrounding environment. Although multiple suspension systems are applied to the detector, we cannot get rid of all these noises. The noise will cause a movement or rotation to that affected component. In an optical system, when a small tilting angle is applied to one of the mirrors within the system, a slight change will be caused to the signal of the Gaussian beam. Figure 1.1 below shows a pair of Gaussian beam plots. The plot on the left side is a simple beam profile of a normal Gaussian beam. The plot on the right side is another Gaussian beam cut with a tilting angle applied.

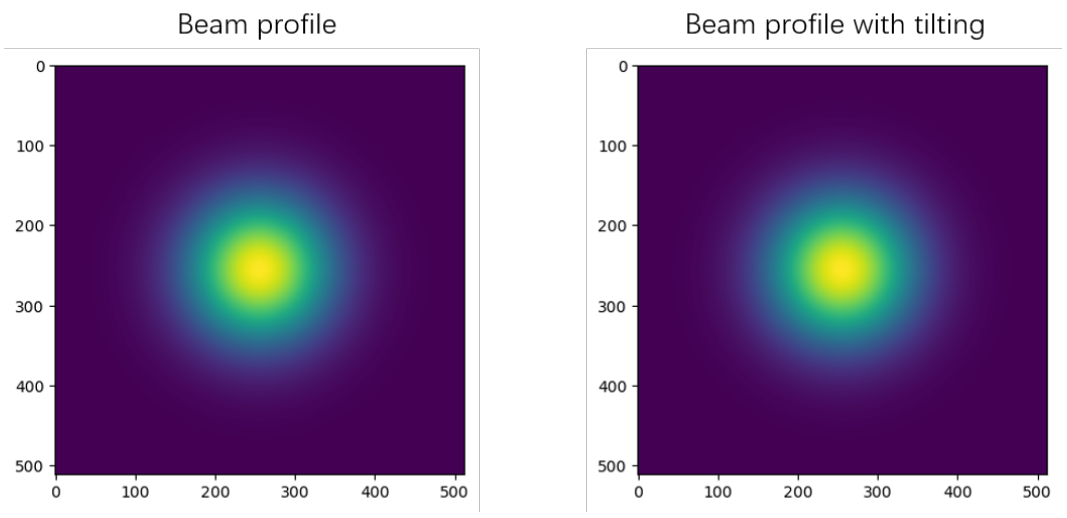

Figure 1.1.: Comparison between two Gaussian beam

Since one can hardly tell the difference between these two beam profiles, it is essential to plot a differential signal of the two beams. We then obtain the following plot:

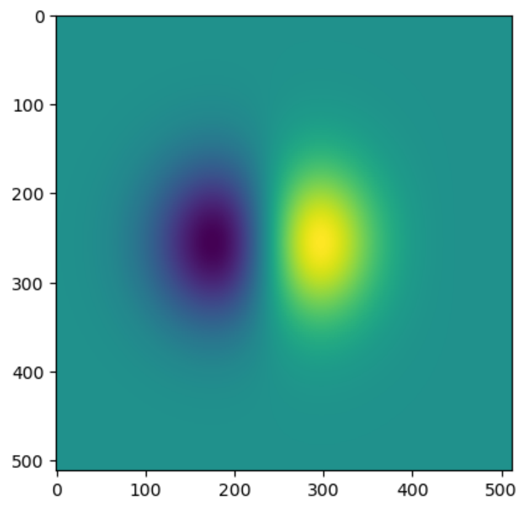

Figure 1.2.: Difference between two cuts

Clearly, with the tilting angle applied to the mirror, it causes a change in the signal. In tradition, we solve such problem by calculating the movement or rotation with the wave-front sensing method. Theoretically, the radiation pressure noise can be calculated by computer in real time with traditional PDH locking and WFS method [4]. However, the mirror is not perfectly smooth in reality, and the centre of the beam can be varied through each operation period. These factors will raise complexity of the calculation and the signal will be distorted. Therefore, we need to find a better way to avoid the offset problem. Recently, deep learning provides us a new method to calculate much complex problems, especially in area with large data like astrophysics. For example, the classification of binary star systems. Thus, we choose to use deep learning models as the automatic control system.

The purpose of this research is to build an automatic control system that react in real time. Previous study from our laboratory shows that deep learning is useful in developing such an automatic control system [5]. A convolutional neural network (CNN) has been built to analyse WFS signal from a Fabry-Perot cavity. The loss of that deep learning model can be reduced to a reasonable range. Based on that conclusion, we also choose to use a deep learning model to help us do the calculation. Another study from our laboratory shows the possibility of using linear combination to build a control system for mirror tilting [6]. However, a linear combination still cannot solve the problem of distortion in signals.

The whole research contains large amount of simulations and several models with different feature. All of them are built in Python language with the help of several related packages. All packages used in this study works in Windows. Therefore, the outcome can be easily reproduced with a proper environment. The whole study can be divided into three parts: generating data, data analysis and deep learning. We will give a brief introduction on each part in the following paragraph.

In the first part, we use an interferometer simulation software called Finesse [7]. Finesse is originally written in C language, with the help of a wrapping package called PyKat [8], we can run this simulation software in Python for later usage. We used Finesse 2 in this study for it is the stable version when the study is started. The newest version, Finesse 3, has been updated recently and is able to run the same simulation with a faster speed. The set up for the simulation is based on a simple Fabry-Perot cavity. A schematic of the set-up is shown in Figure 1.3. A mirror map is applied on its front mirror to simulated complex situations. And a tilting angle is applied on its end mirror to change the output signals. We collect the 2-dimensional DC signal on the reflected side of the cavity by adding a beam detector. For gravitational wave detection, we usually use photo-detectors which contains only 1-dimensional feedback. We are expecting adding multiple photo-detectors to form a array that can simulate a 2-dimensional signals. Therefore, the resolution of the beam detector needs to be restricted and here we use a 4×4 sample rate. When the 4×4 sample rate is set to the beam detector, the simulation returns 16 intensity values as the output of one run. For comparison, a few data sets with different precision and different mirror types are generated, each data set contains 10000 samples with 100 to 10000 cuts, depends on the number of mirror maps. To clarify, these outputs are called "raw data sets".

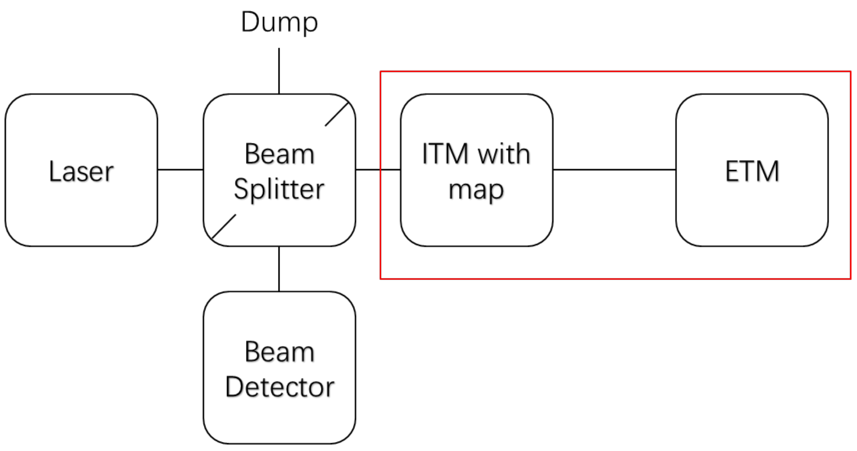

Figure 1.3.: Schematic of the Set-up

In the second part, PCA technique is applied on those raw data sets to reduce the redundant information in it. For an example, with a tiny tilting angle applied to the cavity, the intensity from the corner of a beam detector will not change much. Thus, even though the intensity value from the corner should be taken into training, it should not have the same weight as the intensity value from the centre of the beam detector. PCA technique will help us rearrange the weight of these raw data sets and it will return another 16 values with reorganized basis, we call them "transferred data sets". These transferred data sets are then split into several matrices so that they can be put into the deep learning model and are ready for training.

In the third part, we put the data sets into training. There are several Python packages that provide deep learning related functions. Tensorflow and PyTorch [9] are two of the most famous choice. We chose the latter one for its user-friendly features. In this study, we built four different deep learning models based on two groups of features. The model is either a linear model or with a convolutional neural network structure. And the model is either a classification model or a regression model. When all these models are fully trained by transferred data sets, we can find out the best model and optimize it for later research.

Though the whole study is set on computer simulations, it is necessary to understand the basic ideas of gravitational wave and its detection. The thesis is organized as follows: in the first two chapters, we will discuss some physics theory used in this study. In chapter 2, we will first explain the basic ideas of gravitational waves, how we detect them and the noises of a gravitational wave detector. And in chapter 3, some definitions and calculations from laser physics will be introduced to help understanding the basic physics within the gravitational wave detectors. In chapter 4, there is a detailed introduction of all deep learning related ideas in this study. We will explain everything from the basic neurons to the structure of the whole models. Finally, in chapter 5, the methods and detailed simulation settings will be shown. There is a explanation for the whole simulation system in this chapter. The result of this study will also included in the same chapter. A summary will be written in chapter 6. Some future works will also be contained in the same chapter.

Chapter 2

Gravitational wave

To figure out what how those noises affect on a gravitational detector, we need to understand what are gravitational waves look like and how are they detected. Basic theories like general relativity and the Einstein equation are necessary. In this chapter, we will first introduce the basic physics of general relativity and gravitational waves. Then, we will look back into the development of gravitational detector to see the ideas of gravitational wave detection, which also shows how Fabry-Perot cavity is introduced in the system. The chapter will end up with the main problem of this study, which is the noises in gravitational wave detection.

2.1 General relativity

Back into 1905, Einstein created special relativity. Based on the view of special relativity, Einstein steps further into the universe and created the theory of general relativity. General relativity is a theory that combines time, space and gravity together. It explains the cause of gravity as a kind of curvature in the time-space. And first developed an idea that gravity is not a normal force, but a geometric effect. Einstein also predicted the existence of gravitational waves based on this geometric effect [10]. For decades, this area of study stayed silenced for there is no experimental progress based general relativity. Until Jocelyn Bell Burnell detected the first pulsar star in 1967, people started to put more attention to general relativity and gravitational waves. Finally in 2015, LIGO directly detected the first gravitational wave signal in human history [11]. This section will start with an introduction to general relativity theory.

Gravitational waves are generated by gravitational events and propagate through the space. And its one of the key conclusions of Einstein field equations. For any points in space-time, we have:

$$x^\mu = (-ct, x, y, z) \tag{2.1}$$

Which contains the information of both time and space position. If we consider a slight displacement dx^μ from the point, we have the distance between these two points as:

$$ds^2 = g_{\mu\nu} dx^\mu dx^\nu \tag{2.2}$$

Where $g_{\mu\nu}$ is the metric tensor that represent the structure of space-time near point x^μ. For a flat universe with $\Omega = 1$, it can be represented as the tensor as follows:

$$g_{\mu\nu} = \eta_{\mu\nu} = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.3}$$

More generally, the metric tensor obeys Einstein field equations:

$$G_{\mu\nu} = R_{\mu\nu} - \frac{1}{2} g_{\mu\nu} R \tag{2.4}$$

$$= \frac{8\pi G}{c^4} T_{\mu\nu} \tag{2.5}$$

On the left-hand side, $G_{\mu\nu}$ is the Einstein tensor which represent the status of space and time. On the right-hand side, Energy-momentum tensor $T_{\mu\nu}$ shows the status of matters. G and c in the equation are just constant of gravitation and speed of light. $R_{\mu\nu}$ and R are called Ricci tensor and Ricci scalar respectively, and their definitions are as follow:

$$R_{\mu\nu} = R^\lambda_{\mu\lambda\nu} \tag{2.6}$$

$$R = R^\mu_\mu = g^{\mu\nu} R_{\mu\nu} \tag{2.7}$$

$R^\lambda_{\mu\lambda\nu}$ here is a specific Riemann curvature tensor with general definition as follows:

$$R^\gamma_{\mu\rho\nu} = \frac{\partial \Gamma^\gamma_{\mu\nu}}{\partial x^\rho} - \frac{\partial \Gamma^\gamma_{\mu\rho}}{\partial x^\nu} + \Gamma^\gamma_{\alpha\rho}\Gamma^\alpha_{\mu\nu} - \Gamma^\gamma_{\beta\nu}\Gamma^\beta_{\mu\rho} \tag{2.8}$$

Where Γ is called Christoffel symbol:

$$\Gamma^\rho_{\mu\nu} = \frac{1}{2} g^{\rho\delta} \left(\frac{\partial g_{\nu\delta}}{\partial x^\mu} + \frac{\partial g_{\mu\delta}}{\partial x^\nu} - \frac{\partial g_{\mu\nu}}{\partial x^\delta} \right) \tag{2.9}$$

To simplify, the partial derivative later in this thesis will be written as the following notation:

$$A_{\mu,\nu} := \frac{\partial}{\partial x^\nu} A_\mu \tag{2.10}$$

2.2 Solution of Einstein Equation

For Minkowski space where no matter exists, the right-hand side of Einstein Field Equation is set to 0 because of the Energy-momentum $T_{\mu\nu}$ is 0 . Which gives:

$$G_{\mu\nu} = R_{\mu\nu} - \frac{1}{2} g_{\mu\nu} R = 0 \tag{2.11}$$

If we consider a small perturbation in flat Minkowski space, we have the metric tensor as:

$$g_{\mu\nu} = \eta_{\mu\nu} + h_{\mu\nu} \tag{2.12}$$

The Christoffel symbol are recalculated as:

$$\begin{aligned}
\Gamma^{\rho}_{\mu\nu} &= \frac{1}{2}\left(\eta^{\rho\delta} + h^{\rho\delta}\right)\left(\frac{\partial\left(\eta_{\nu\delta} + h_{\nu\delta}\right)}{\partial x^{\mu}} + \frac{\partial\left(\eta_{\mu\delta} + h_{\mu\delta}\right)}{\partial x^{\nu}} - \frac{\partial\left(\eta_{\mu\nu} + h_{\mu\nu}\right)}{\partial x^{\delta}}\right) \\
&= \frac{1}{2}\left(\eta^{\rho\delta} + h^{\rho\delta}\right)\left(h_{\nu\delta,\mu} + h_{\mu\delta,\nu} - h_{\mu\nu,\delta}\right) \\
&\approx \frac{1}{2}\eta^{\rho\delta}\left(h_{\nu\delta,\mu} + h_{\mu\delta,\nu} - h_{\mu\nu,\delta}\right) \qquad (2.13)
\end{aligned}$$

Ricci tensor and Ricci scalar are represented as:

$$\begin{aligned}
R_{\mu\nu} &= R^{\lambda}_{\mu\lambda\nu} \\
&= \frac{\partial\Gamma^{\lambda}_{\mu\nu}}{\partial x^{\lambda}} - \frac{\partial\Gamma^{\lambda}_{\mu\lambda}}{\partial x^{v}} + \Gamma^{\lambda}_{\alpha\lambda}\Gamma^{\alpha}_{\mu\nu} - \Gamma^{\lambda}_{\beta\nu}\Gamma^{\beta}_{\mu\lambda} \\
&\approx \frac{\partial\frac{1}{2}\eta^{\lambda\delta}\left(h_{\nu\delta,\mu} + h_{\mu\delta,\nu} - h_{\mu\nu,\delta}\right)}{\partial x^{\lambda}} - \frac{\partial\frac{1}{2}\eta^{\lambda\delta}\left(h_{\lambda\delta,\mu} + h_{\mu\delta,\lambda} - h_{\mu\lambda,\delta}\right)}{\partial x^{\nu}} \\
&= \frac{1}{2}\eta^{\lambda\delta}\left(h_{\nu\delta,\mu\lambda} + h_{\mu\delta,\nu\lambda} - h_{\mu\nu,\delta\lambda}\right) \qquad (2.14) \\
R &= g^{\mu\nu}R_{\mu\nu} \\
&\approx \frac{1}{2}\eta^{\mu\nu}\eta^{\lambda\delta}\left(h_{\nu\delta,\mu\lambda} + h_{\mu\delta,\nu\lambda} - h_{\mu\nu,\delta\lambda}\right) \qquad (2.15)
\end{aligned}$$

The Einstein equation can be rewritten as:

$$\begin{aligned}
G_{\mu\nu} &= R_{\mu\nu} - \frac{1}{2}g_{\mu\nu}R \\
&= \frac{1}{2}\left[h^{\delta}_{\nu,\mu\delta} + h^{\delta}_{\mu,\nu\delta} - \Box h_{\mu\nu} - h_{,\mu\nu} - \eta_{\mu\nu}\left(h^{\delta\sigma}_{,\delta\sigma} - \Box h\right)\right] \qquad (2.16)
\end{aligned}$$

With d'Alembert operator $\Box = \partial^{\mu}\partial_{\mu} = \eta^{\mu v}\partial_{\mu}\partial_{\nu}$. Then if we rewrite the perturbation term and define the trace-reversed tensor as:

$$\tilde{h}_{\mu\nu} = h_{\mu\nu} - \frac{1}{2}\eta_{\mu\nu}h \tag{2.17}$$

We can simplify Einstein equation as:

$$G_{\mu\nu} = \frac{1}{2}\left(\tilde{h}^{\delta}_{\nu,\mu\delta} + \tilde{h}^{\delta}_{\mu,\nu\delta} - \Box\tilde{h}_{\mu\nu} - \eta_{\mu\nu}\tilde{h}^{\delta\sigma}_{,\delta\sigma}\right) \tag{2.18}$$

Now consider the following Lorenz gauge conversion:

$$x'^{\mu} = x^{\mu} + \xi(x) \tag{2.19}$$

$$\begin{aligned} g'_{\mu\nu} &= \frac{\partial x^{\alpha}}{\partial x'^{\mu}}\frac{\partial x^{\beta}}{\partial x'^{\nu}} g_{\alpha\beta} \\ &= \left(\delta^{\alpha}_{\mu} - \xi^{\alpha}_{,\mu}\right)\left(\delta^{\beta}_{\nu} - \xi^{\beta}_{,\nu}\right)\left(\eta_{\alpha\beta} + h_{\alpha\beta}\right) \\ &= \eta_{\mu\nu} + h_{\mu\nu} - \xi_{\mu,\nu} - \xi_{\nu,\mu} \end{aligned} \tag{2.20}$$

The perturbation term of Lorenz gauge conversion is:

$$h'_{\mu\nu} = g'_{\mu\nu} - \eta_{\mu\nu} = h_{\mu\nu} - \xi_{\mu,\nu} - \xi_{v,\mu} \tag{2.21}$$

$$h' = h - \xi^{\sigma}_{,\sigma} - \xi^{\sigma}_{,\sigma} = h - 2\xi^{\sigma}_{,\sigma} \tag{2.22}$$

The trace-reversed tensor becomes:

$$\begin{aligned} \tilde{h}'_{\mu\nu} &= h'_{\mu\nu} - \frac{1}{2}\eta_{\mu\nu}h' \\ &= h_{\mu\nu} - \xi_{\mu,\nu} - \xi_{\nu,\mu} - \frac{1}{2}\eta_{\mu\nu}\left(h - 2\xi^{\sigma}_{,\sigma}\right) \\ &= \tilde{h}_{\mu\nu} - \xi_{\mu,\nu} - \xi_{\nu,\mu} + \eta_{\mu\nu}\xi^{\sigma}_{,\sigma} \end{aligned} \tag{2.23}$$

We can always find one ξ^{μ} that satisfies the 1st derivative of Lorenz gauge conversion to be 0 $\left(\tilde{h}'^{\mu}_{\nu,\mu} = 0\right)$. Under this harmonic condition, all Lorenz gauge term can be omitted and we simplify Einstein's equation as:

$$G_{\mu\nu} = -\frac{1}{2}\Box\tilde{h}_{\mu\nu} \tag{2.24}$$

The Gravitational wave equation constrains 2nd derivative of $h_{\mu\nu}$ through $\tilde{h}_{\mu\nu}$:

$$\Box\tilde{h}_{\mu\nu} = -\frac{16\pi G}{c^4}T_{\mu\nu} \tag{2.25}$$

Since in flat universe, we have the energy momentum tensor to be 0 , the gravitational wave equation can be simplified to:

$$\Box\tilde{h}_{\mu\nu} = 0 \tag{2.26}$$

Assume we have a plane wave as:

$$\tilde{h}_{\mu\nu} = A_{\mu\nu} e^{i\left(k_\lambda x^\lambda - \phi_0\right)} \tag{2.27}$$

We figure out:

$$\Box A_{\mu\nu} e^{i\left(k_\lambda x^\lambda - \phi_0\right)} = A_{\mu\nu} \eta^{\lambda\sigma} \partial_\lambda \partial_\sigma e^{i\left(k_\lambda x^\lambda - \phi_0\right)} = 0 \tag{2.28}$$

For that, we only need $\eta^{\lambda\sigma} k_\lambda k_\sigma = 0$. Lorenz and transverse trace-less gauges also constrain $A_{\mu\nu}$:

$$k^\mu A_{\mu\nu} = 0, \quad A_{\mu\nu} u^\nu = 0, \quad A^\mu_\mu = 0 \tag{2.29}$$

Which gives us:

$$ck^0 = k^3, \quad A_{\mu 0} = A_{\mu 3} = 0, \quad A_{22} = -A_{11} \tag{2.30}$$

The plane wave solution simplifies to:

$$\begin{aligned} \tilde{h}_{\mu\nu} &= h_{\mu\nu} \\ &= A_{\mu\nu} e^{i\left(k_\lambda x^\lambda - \phi_0\right)} \\ &= \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & A_{11} & A_{12} & 0 \\ 0 & A_{12} & -A_{11} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} e^{i\left(k^3(z-ct) - \phi_0\right)} \end{aligned} \tag{2.31}$$

Therefore, factor $A_{\mu\nu}$ can be polarized into two basis tensors:

$$h_{\mu\nu} = \left(A_+ h^+_{\mu\nu} + A_\times h^\times_{\mu\nu}\right) e^{-i(\phi_0 + \omega t)} \tag{2.32}$$

Where:

$$A_+ \equiv A_{11}, \quad A_\times \equiv A_{12} \tag{2.33}$$

$$h^+_{\mu\nu} \equiv \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad h^\times_{\mu\nu} \equiv \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.34}$$

These represent the two gravitational wave polarization, usually referred to as ”plus” and ”cross” gravitational waves. A schematic of these two polarization are as follows[12]:

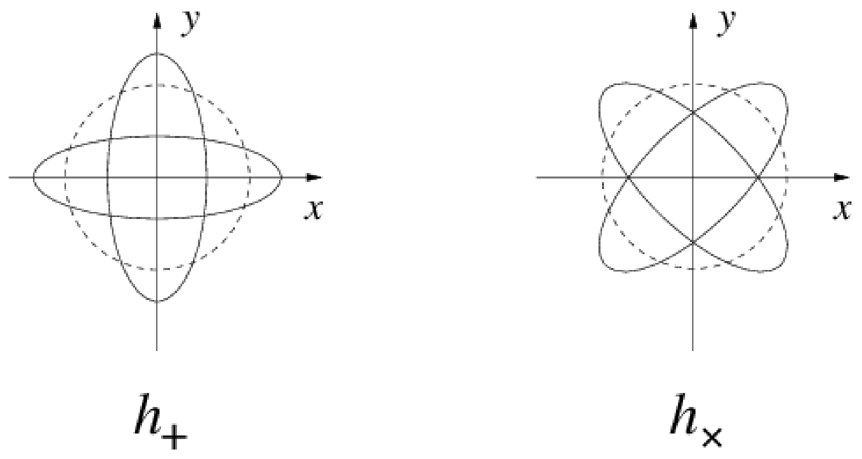

Figure 2.1.: The two gravitational wave polarization

2.3 Gravitational wave detectors

Since Einstein first predicted the existence of gravitational waves. Various type of gravitational wave detectors has been developed to fetch a real gravitational wave signal. Around 1968, Joe Weber first started the search for gravitational waves. His resonant mass detectors are developed and built at the university of Maryland, called Weber bar [13]. It consisted of huge aluminium cylinders with the total size of 2 meters in length and 1 meter in diameter. The aluminium is separated into two parts, each end is like a test mass, while the centre is like a spring. He assumed that a gravitational wave excites vibrational motion. And the strain gauges at the mid-line could sense that vibration. Though Weber claimed that one gravitational wave has been detected, he cannot reproduce any other outcome like that event. The good thing is, Weber's effort still motivated the researchers in this area of study. His study became a world-wide effort to detect gravitational waves and to develop related technologies. Interferometer groups started to be developed at Caltech, MIT and other universities. Eventually, the idea of building a km-scale gravitational wave detector was put into practice.

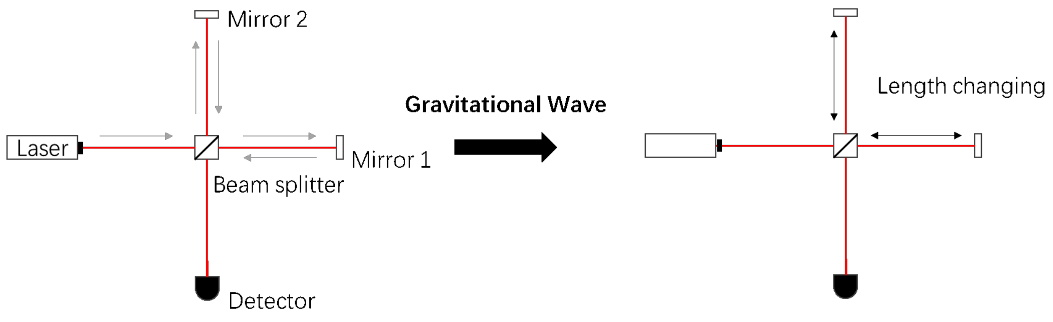

Figure 2.2.: Detection of gravitational waves

Basically, a ground-based gravitational wave detector is a Michelson interferometer with a larger scale. Figure 2.1 shows how a gravitational wave be detected by an interferometer detector. The laser comes from the left first hits a beam splitter. Then the beam splitter divides the laser into two beams. Each beam then goes through a long tunnel, hits the mirror and reflected back to the beam splitter. Two beams combine as one signal at this point and the Detector collects the reflected signal.

Assume we have the initial laser as E_{in}. It will be split by the beam splitter as follows:

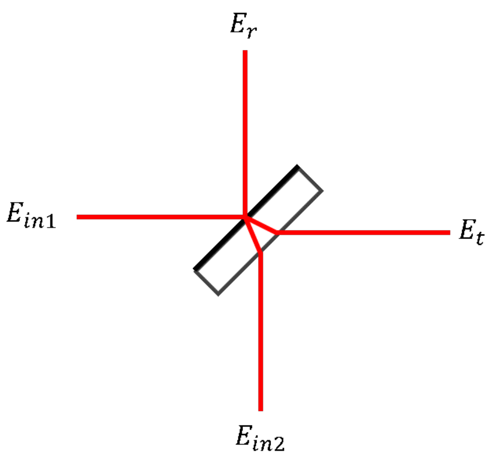

Figure 2.3.: Beam splitter

For a 50:50 lossless beam splitter:

$$E_r = \tilde{r}_{\text{bs}} E_{\text{in}} \tag{2.35}$$

$$E_t = \tilde{t}_{\text{bs}} E_{\text{in}} \tag{2.36}$$

$$|\tilde{r}_{\text{bs}}| = \left|\tilde{t}_{\text{bs}}\right| = \frac{1}{\sqrt{2}} \tag{2.37}$$

The interference is the sum of multiple waves. We have the transmitted wave and reflected wave as:

$$E_r = \frac{1}{\sqrt{2}} E_{\text{in1}} + \frac{1}{\sqrt{2}} E_{\text{in2}} \tag{2.38}$$

$$E_t = \frac{1}{\sqrt{2}} E_{\text{in1}} - \frac{1}{\sqrt{2}} E_{\text{in2}} \tag{2.39}$$

Assuming beam splitter phase relation I on transmission, we have:

$$\begin{aligned} E_r &= \frac{1}{\sqrt{2}} E_{\text{in1}} \frac{1}{\sqrt{2}} e^{\frac{2i\omega L_2}{c}} + \frac{1}{\sqrt{2}} E_{\text{in2}} \frac{1}{\sqrt{2}} e^{\frac{2i\omega L_1}{c}} \\ &= -i \sin\left(\frac{2\omega \Delta L}{c}\right) E_{\text{in}} e^{\frac{2i\omega L}{c}} \end{aligned} \tag{2.40}$$

Where $L_1 = L + \Delta L$ and $L_2 = L - \Delta L$. Given that $P = E^* E$, we can calculate the laser power changes as:

$$\frac{P_r}{P_{\text{in}}} = \frac{1 - \cos \frac{4\omega \Delta L}{c}}{2} \tag{2.41}$$

And the maximum and minimum power of signal are respectively:

$$P_{\text{max}} \equiv (E_r + E_t)^2 \tag{2.42}$$

$$P_{\text{min}} \equiv (E_r - E_t)^2 \tag{2.43}$$

Remember we have discussed the propagation of gravitational waves along z-axis with two different modes, the plus wave and the cross wave. When a gravitational wave hits from the top of a Michelson interferometer, it will cause a phase change in its two arms. Thus, we can detect the gravitational wave from the interference pattern or the interferometer.

The first direct observation of gravitational waves has been made by LIGO and Virgo in 2015. Nowadays we even have multiple ground-based interferometer detectors that works together, and indeed we have a lot more advantages than one single detector. We currently have aLIGO (Advanced LIGO) in Hanford and Livingston in United States, Virgo in Europe, and KAGRA [14] in Japan working together as one whole system. And this study is based on the noise problems in this type of detectors.

2.4 Noises of gravitational wave detectors

We know the gravitational wave is so weak that resonant mass detectors failed to detect it directly. A plot of estimated signal strengths for both the interferometer detectors and other detectors is as follows:

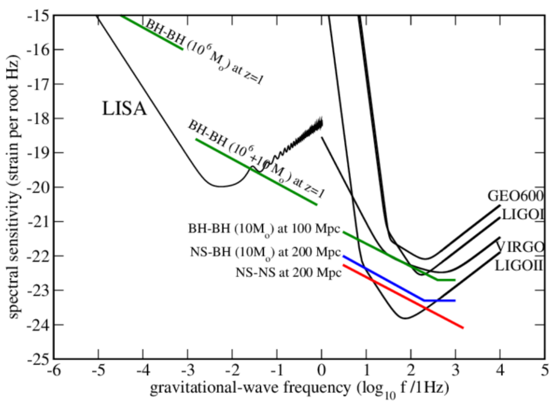

Figure 2.4.: Estimated signal strengths for various detectors

For interferometer detectors like LIGO, noises are sill one of the main problems for a good observation. In this section, we will introduce some of the main noises in gravitational wave detection. They are shot noise, radiation pressure noise, thermal noise and seismic noise.

As long as we use laser beam as our tool, we cannot avoid shot noise. The cause of shot noises is the quantum fluctuation of photon number in the laser. To be simple, shot noise is the error of photon counting from the photon detector. Shot noises can be reduced by increasing the intensity of the laser. However, this method also causes other problems such as the radiation pressure noise.

As the laser passes through the interferometer and hits the mirror and other optical equipment, the photon within the laser will lose a part of its energy to the mirror, causing the mirror to move back and forth. Such movement is called radiation pressure noise. Unlike shot noise, radiation pressure noise will raise as the laser intensity raises. This is why we cannot handle these two noises at the same time. The balance between these two noises gives us a limitation we can reach. This limit is called is called standard quantum limit (SQL). Although this limitation is the mechanical limit of lasers, technologies such as quantum squeezing are developed and used to break through this limitation [15, 16].

Factors of the global system also affect the detector. As temperature of the system changes, fluctuation appears on both the mirror and other equipment like the suspension systems of the experiment, which causes mirror thermal noise. The vibration of the equipment itself also moves as a pendulum, which causes suspended thermal noise. To avoid these kinds of noises, it is better to choose an experiment environment with lower temperature. For example, KAGRA attempted to lower the temperature of the whole tunnel to reduce thermal noises.

By far, all these interferometers are ground-based detectors. The seismic noise is an unavoidable problem. Earth vibration are the main cause of this kind of noise. For underground detector like KAGRA, underwater flows also become one of the main reasons of seismic noise. To avoid this kind of noise, we usually apply multiple suspension system on the mirrors. There is also another approach to develop a space-based detector, called LISA. The pathfinder of LISA program has been launched since 2015.

In this study, we focus on the noises that will cause a displacement to the optical system, such as seismic noises and radiation pressure noise. The automatic control system we develop shall be able to recognize the displacement of the mirror and see if we can add another movement to counter the radiation pressure noise.

Chapter 3

Laser Physics

To build an automatic control system against radiation pressure noise, it would be too complex to analyse the whole interferometer detector in one model. Therefore, our aim is to build a simple optical system just like what previous study did, and make one step further. In this chapter, we will start with a brief review on the features of Gaussian beam. And we will end up with an explanation on the simple optical system we choose, which is the Fabry-Perot cavity.

3.1 Gaussian Beam

In optics, a Gaussian beam is an electromagnetic wave beam whose electric field on transverse direction and intensity distribution approximately satisfy a Gaussian function. In experiment, many lasers can be approximately treated as a Gaussian beam. When it is refracted in a near-diffraction-limited lens, a Gaussian beam is transformed into another Gaussian beam with different beam parameters. Therefore, a Gaussian beam is a convenient and widely used model in optical systems. The wave equation of a particular Gaussian beam can be found by Maxwell's Equation in electromagnetism. The first step is to find out the equation for electromagnetic waves. Remember we have Maxwell's Equation with vacuum condition as follows:

$$\nabla \cdot E = 0 \tag{3.1}$$

$$\nabla \cdot B = 0 \tag{3.2}$$

$$\nabla \times E = -\frac{\partial B}{\partial t} \tag{3.3}$$

$$\nabla \times B = \frac{1}{c^2}\frac{\partial E}{\partial t} \tag{3.4}$$

If we take the curl on Maxwell-Faraday equation, we have:

$$\nabla \times (\nabla \times E) = -\frac{\partial}{\partial t}\nabla \times B \tag{3.5}$$

We can replace the terms with Gauss's law and Ampere's circuital law:

$$\nabla \times (\nabla \cdot E) - \nabla^2 E = -\frac{\partial}{\partial t}\frac{1}{c^2}\frac{\partial E}{\partial t} \tag{3.6}$$

$$-\nabla^2 E = -\frac{1}{c^2}\frac{\partial^2 E}{\partial t^2} \tag{3.7}$$

Rearrange the equation and we have the wave equation as follows:

$$\left(\nabla^2 - \frac{1}{c^2}\frac{\partial^2}{\partial t^2}\right) E = 0 \tag{3.8}$$

To solve this equation, assume we have the following solution for a plane wave:

$$E(x,t) = u(x,y,z)e^{i(\omega_0 t - kz)} \tag{3.9}$$

The wave propagates along z-axis. The real part of this solution represents the electric field, while ω_0 hear is the angular frequency, k is the wave number defined by $k = \frac{\omega_0}{c}$, and u is called displacement vector, which represents the displacement from the plane. We take this solution back into the wave equation and we will find:

$$\begin{aligned}
\left(\nabla^2 - \frac{1}{c^2}\frac{\partial^2}{\partial t^2}\right) E &= \left(\nabla^2 - \frac{1}{c^2}\frac{\partial^2}{\partial t^2}\right) u(x,y,z)e^{i(\omega_0 t - kz)} \\
&= \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} - \frac{1}{c^2}\frac{\partial^2}{\partial t^2}\right) u(x,y,z)e^{i(\omega_0 t - kz)} \\
&= \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) u(x,y,z)e^{i(\omega_0 t - kz)} \\
&\quad + \frac{\partial}{\partial z}\left[\frac{\partial}{\partial z}u(x,y,z) - iku(x,y,z)\right] e^{i(\omega_0 t - kz)} \\
&\quad + k^2 u(x,y,z)e^{i(\omega_0 t - kz)} \\
&= \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) u(x,y,z)e^{i(\omega_0 t - kz)} \\
&\quad + \left(\frac{\partial^2}{\partial z^2}u(x,y,z) - 2ik\frac{\partial}{\partial z}u(x,y,z)\right) e^{i(\omega_0 t - kz)} \\
&= \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} - 2ik\frac{\partial}{\partial z}\right) u(x,y,z)e^{i(\omega_0 t - kz)}
\end{aligned} \tag{3.10}$$

Therefore, the condition is:

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} - 2ik\frac{\partial}{\partial z}\right) u(x,y,z) = 0 \tag{3.11}$$

Furthermore, we assume the wave does not change along z-axis to make it simpler. We take $\frac{\partial^2 u}{\partial z^2} \approx 0$ and the condition is simplified to:

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} - 2ik\frac{\partial}{\partial z}\right) u(x, y, z) = 0 \tag{3.12}$$

This is called the par-axial approximation of Helmholtz equation. We can assume a general solution for it as follows:

$$u = A(z)e^{-i\frac{k}{2q(z)}(x^2+y^2)} \tag{3.13}$$

If we take this solution back into the left-hand side of par-axial approximation of Helmholtz equation, we have:

$$\begin{aligned}
&\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} - 2ik\frac{\partial}{\partial z}\right) A(z)e^{-i\frac{k}{2q(z)}(x^2+y^2)} \\
&= A(z)\frac{\partial^2}{\partial x^2}e^{-i\frac{k}{2q(z)}(x^2+y^2)} + A(z)\frac{\partial^2}{\partial y^2}e^{-i\frac{k}{2q(z)}(x^2+y^2)} - A(z)2ik\frac{\partial}{\partial z}e^{-i\frac{k}{2q(z)}(x^2+y^2)} \\
&= -\frac{k^2}{q^2}A(z)(x^2+y^2)e^{-i\frac{k}{2q(z)}(x^2+y^2)} - 2i\frac{k}{q}A(z)e^{-i\frac{k}{2q(z)}(x^2+y^2)} \\
&\quad - 2ik\frac{dA(z)}{dz}e^{-i\frac{k}{2q(z)}(x^2+y^2)} + \frac{k^2}{q^2}A(z)\frac{dq}{dz}(x^2+y^2)e^{-i\frac{k}{2q(z)}(x^2+y^2)} \\
&= \left[\frac{k^2}{q^2}\left(\frac{dq}{dz} - 1\right)(x^2+y^2) - 2ik\left(\frac{1}{q} + \frac{1}{A}\frac{dA}{dz}\right)\right] A(z)e^{-i\frac{k}{2q(z)}(x^2+y^2)}
\end{aligned} \tag{3.14}$$

To satisfy the par-axial approximation of Helmholtz equation, we find out the following conditions:

$$-1 + \frac{dq}{dz} = 0 \tag{3.15}$$

$$\frac{1}{q} + \frac{1}{A}\frac{dA}{dz} = 0 \tag{3.16}$$

To solve these conditions, we solve first order differential equations for the first condition and we have the equation of $q(z)$ in the following form:

$$q(z) = z + q_0 \tag{3.17}$$

The second condition can be rewrite into following form:

$$\frac{1}{A}\frac{dA}{dz} = -\frac{1}{q} \tag{3.18}$$

$$\log A = -\log q + c \tag{3.19}$$

Since we need to find the relationship between there parameters, we need to fit the boundary conditions into the equation. Simply bring in boundary condition $A(0) = A_0$ and we have:

$$c = \log A_0 q_0 \tag{3.20}$$

Therefore, we can see that $A(z)$ is a function of $q(z)$. We now take boundary condition into consideration. We assume the beam intensity μ goes to 0 when it is far from the beam centre $r^2 = x^2 + y^2 \to \infty$, which means:

$$\begin{aligned} |\mu|^2 &\propto e^{-i\frac{k}{2q(z)}r^2} e^* e^{-i\frac{k}{2q(z)}r^2} \\ &= e^{-i\frac{k}{2}r^2\left(\frac{1}{q(z)} - \frac{1}{q(z)^*}\right)} \\ &= e^{-i\frac{k}{2q(z)^2}r^2(q(z)^* - q(z))} \\ &= e^{-\frac{k}{q(z)^2}r^2 \operatorname{Im}(q(z))} \end{aligned} \tag{3.21}$$

Thus, to satisfy the boundary condition, we need the imaginary part of $q(z)$ to be positive. We see that q_0 is related to the movement of the beam along z-axis. If we take the centre of the beam of $q(z)$ and note it with z_0. And we can also note the imaginary part of $q(z)$ as z_R, we can have the following equation:

$$q_0 = -z_0 + iz_R \tag{3.22}$$

$$q(z) = (z - z_0) + iz_R \tag{3.23}$$

We now have a full view on the shape of the beam:

$$u(r, z) = A_0 \frac{-z_0 + iz_R}{(z - z_0) + iz_R} e^{-i\frac{k}{2(z-z_0)+2iz_R}r^2} \tag{3.24}$$

The shape of the beam can be determined by the two variables above. z_0 defines the centre of the beam, called the beam waist. While z_R defines the divergence rate of the beam, called Rayleigh range. z_R can also be rewritten as follows:

$$z_R = \frac{kw_0^2}{2} \tag{3.25}$$

Furthermore, if we look back into the parameter of the beam. We can expand the parameter as follows:

$$\begin{aligned}\frac{-z_0+iz_R}{(z-z_0)+iz_R} &= \frac{iz_R\left(z-z_0-iz_R\right)}{\left(z-z_0-iz_R\right)\left(z-z_0+iz_R\right)} \\ &= \frac{z_R}{\left(z-z_0\right)^2+z_R^2}\left[z_R+i\left(z-z_0\right)\right] \\ &= \frac{z_R}{\left(z-z_0\right)^2+z_R^2}\sqrt{\left(z-z_0\right)^2+z_R^2}\,e^{i\arctan\frac{z-z_0}{z_R}} \\ &= \frac{1}{\sqrt{\left(\frac{z-z_0}{z_R}\right)+1^2}}e^{i\arctan\frac{z-z_0}{z_R}} \qquad (3.26)\end{aligned}$$

We can also expand the exponential part of the beam as follows:

$$\begin{aligned}e^{-i\frac{k}{2(z-z_0)+2i_R}r^2} &= e^{-i\frac{1z_R\left(z-z_0-iz_R\right)}{w_0^2(z-z_0)^2+z_R^2}r^2} \\ &= e^{-\frac{1}{w_0^2}\frac{1+i\frac{z}{z_R}}{1+\left(\frac{z-z_0}{z_R}\right)^2}r^2} \qquad (3.27)\end{aligned}$$

Bringing these two expansions back into the beam equation and we have

$$u(r,z)=A_0\frac{1}{\sqrt{\left(\frac{z-z_0}{z_R}\right)+1^2}}e^{i\arctan\frac{z-z_0}{z_R}-\frac{1}{w_0^2}\frac{1+i\frac{z}{z_R}}{1+\left(\frac{z-z_0}{z_R}\right)^2}r^2} \qquad (3.28)$$

To simplify the beam equation, we define the following new parameters:

$$w^2(z)=w_0^2\left[1+\left(\frac{z-z_0}{z_R}\right)^2\right] \qquad (3.29)$$

$$R(z)=(z-z_0)\left[1+\left(\frac{z_R}{z-z_0}\right)^2\right] \qquad (3.30)$$

$$\eta(z)=\arctan\frac{z-z_0}{z_R} \qquad (3.31)$$

Here, $\omega(z)$ is called spot size, which represent radius of them beam at the beam waist z_0. $R(z)$ is radius of curvature, which represent how the beam get diverse. And $\eta(z)$ is called Gouy phase of the beam. Gouy phase is a dimensionless value which contains higher order information of the beam.

Remember that we already have beam waist z_0 and Rayleigh range z_R. We now have altogether five beam parameters determined, and the general form of beam equation is as follows:

$$u(r,z) = A_0 \frac{w_0}{w} e^{-\left(\frac{1}{w^2}+i\frac{k}{2R}\right)r^2+i\eta} \tag{3.32}$$

3.2 Beam Parameters

Next, we will explain the physical meaning of those five beam parameters in details: spot size, beam waist Rayleigh range, radius of curvature and Gouy phase. Here is a simple schematic figure for a Gaussian beam from the side:

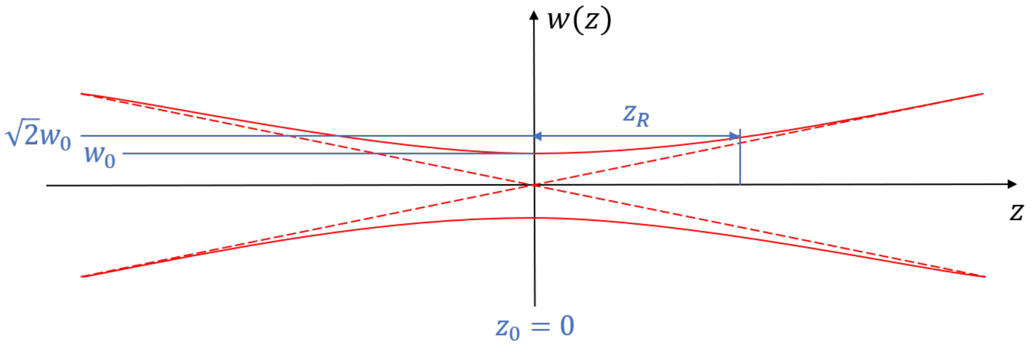

Figure 3.1.: Gaussian Beam

3.2.1 Spot Size $w(z)$

Spot size of the beam shows the spread of the beam along xy-plane. The intensity of the beam is:

$$I(r,z) = |u|^2 = A_0^2 \frac{w_0^2}{w^2} e^{-\frac{2r^2}{w^2}} \tag{3.33}$$

Intensity of the beam will gradually decrease as it goes far away from the beam centre. As is shown in the following figure:

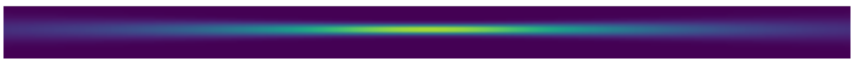

Figure 3.2.: Side view of Gaussian Beam

The power can be calculated as:

$$\begin{aligned} P\left(r_0\right) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I(r,z)dxdy \\ &= \int_0^{2\pi}\int_0^{r_0} I(r,z)rdrd\theta \\ &= 2\pi\int_0^{r_0} I(r,z)rdr \\ &= 2\pi\int_0^{r_0} A_0^2\frac{w_0^2}{w^2}e^{-\frac{2r^2}{w^2}}rdr \\ &= \pi\int_0^{r_0} A_0^2\frac{w_0^2}{w^2}e^{-\frac{s^2}{w^2}}ds \\ &= \frac{\pi}{2}A_0^2w_0^2\left(1-e^{-\frac{2r^2}{\omega^2}}\right) \end{aligned} \tag{3.34}$$

If we calculate the total power towards infinity, we have:

$$P_0 = \frac{\pi}{2}A_0^2w_0^2 \tag{3.35}$$

While the power within the spot size is:

$$P_\omega = \frac{\pi}{2}A_0^2w_0^2\left(1-e^{-2}\right) \tag{3.36}$$

$$= P_0\left(1-e^{-2}\right) \tag{3.37}$$

We can see the area within the spot size contains about 86.4% of total beam power. This can be used to represent the radius of the beam quite well.

3.2.2 Beam waist z_0

Beam waist represent the position where the gaussian beam if most conversed. That is to say, at $z = z_0$, we have the spot size $w(z)$ reaches its minimum. Usually, we also note the spot size at the beam waist as $w_0 = w\left(z_0\right)$. And can be calculated by Rayleigh range as follows:

$$w_0 = \sqrt{\frac{\lambda}{\pi}z_R} \tag{3.38}$$

3.2.3 Rayleigh range z_R

Rayleigh range represents the divergence rate of the beam:

$$z_R = \frac{1}{2}kw_0^2 \tag{3.39}$$

By definition, it is the distance along the propagation direction of a beam from the beam waist to the place where the spot size is doubled:

$$w\left(z_0 \pm z_R\right)=\sqrt{2} w_0 \tag{3.40}$$

Therefore, a large Rayleigh range means the beam will extend a longer distance before its spot size get doubled, which also means that the beam would stay converged for a longer distance in total.

3.2.4 Radius of Curvature $R(z)$

Radius of Curvature of a gaussian beam basically has the same physical meaning as it is in mathematics:

$$R(z)=\left(z-z_0\right)\left[1+\left(\frac{z_R}{z-z_0}\right)^2\right] \tag{3.41}$$

It equals to the radius $R(z)$ of the circular arc which best approximates the curve at the very point z. If we take a look at the phase of a gaussian beam:

$$\phi=-\frac{k}{2 R(z)} r^2-k z+\eta(z) \tag{3.42}$$

We can always find a sphere M_ϕ that has the same phase ϕ. If we note z_ϕ as the cross point between z-axis and that sphere, we can simplify the above equation and have the phase as:

$$\phi=-k z_\phi+\eta\left(z_\phi\right) \tag{3.43}$$

Thus, the sphere M_ϕ can be represented as:

$$-k z_\phi+\eta\left(z_\phi\right)=-\frac{k}{2 R(z)} r^2-k z+\eta(z) \tag{3.44}$$

Rearrange this equation and we can have the change of Gouy phase as:

$$\eta(z)-\eta\left(z_\phi\right)=\frac{k}{2 R(z)} r^2+k z-k z_\phi \tag{3.45}$$

We know the Gouy phase a defined by arc-tan function and we have the summation law of arc-tan as follows:

$$\arctan x-\arctan y=\arctan \frac{x-y}{1+x y} \tag{3.46}$$

We also know that for the region near the sphere of same phase, we can assume $\left|z-z_\phi\right| \ll z_R$. Which means we can use the 1^{st} order Maclaurin expansion and simplify the arc-tan term as follows:

$$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} + \cdots \approx x \tag{3.47}$$

Expanding our phase equation with these two assumptions and we have:

$$\begin{aligned}
\eta(z) - \eta\left(z_\phi\right) &= \arctan \frac{z - z_0}{z_R} - \arctan \frac{z_\phi - z_0}{z_R} \\
&= \arctan \frac{\frac{z-z_0}{z_R} - \frac{z_\phi - z_0}{z_R}}{1 + \frac{z-z_0}{z_R}\frac{z_\phi - z_0}{z_R}} \\
&= \arctan \frac{z_R\left(z - z_\phi\right)}{z_R^2 + \left(z - z_0\right)\left(z_\phi - z_0\right)} \\
&\approx \frac{z_R\left(z - z_\phi\right)}{z_R^2 + \left(z - z_0\right)\left(z_\phi - z_0\right)}
\end{aligned} \tag{3.48}$$

For a normal laser well used in optical systems, the wave length λ is about 10^{-7} m. Which is far smaller than the Rayleigh range, which at about 10°m order of magnitude. Therefore, we can assume $\lambda \ll z_R$ and make a simplification as follows:

$$\eta(z) - \eta\left(z_\phi\right) \approx \frac{1}{z_R}\left(z - z_\phi\right) \ll k\left(z - z_\phi\right) \tag{3.49}$$

Therefore, change of the Gouy phase can be omitted when we are calculating near sphere M_ϕ, which gives:

$$\frac{k}{2R(z)} r^2 + k\left(z - z_\phi\right) = 0 \tag{3.50}$$

By the definition, we also have the change of the radius of curvature as:

$$\begin{aligned}
R(z) - R\left(z_\phi\right) &= \left(z - z_0\right)\left[1 + \left(\frac{z_R}{z - z_0}\right)^2\right] - \left(z_\phi - z_0\right)\left[1 + \left(\frac{z_R}{z_\phi - z_0}\right)^2\right] \\
&= \left(z - z_0\right) + \frac{z_R^2}{z - z_0} - \left(z_\phi - z_0\right) + \frac{z_R^2}{z_\phi - z_0} \\
&= z - z_\phi + \frac{z_R^2}{z - z_0} - \frac{z_R^2}{z_\phi - z_0} \\
&= z - z_\phi + z_R^2\left[\frac{z_\phi - z}{\left(z - z_0\right)\left(z_\phi - z_0\right)}\right] \\
&= \left(z - z_\phi\right)\left[1 - \frac{z_R^2}{\left(z - z_0\right)\left(z_\phi - z_0\right)}\right]
\end{aligned} \tag{3.51}$$

Rearrange the equation of sphere M_ϕ and we have:

$$r^2 + 2R(z)\left(z - z_\phi\right) = 0 \tag{3.52}$$

$$r^2 + 2\left[R\left(z_\phi\right) + \left(z - z_\phi\right)\left[1 - \frac{z_R^2}{\left(z - z_0\right)\left(z_\phi - z_0\right)}\right]\right]\left(z - z_\phi\right) = 0 \tag{3.53}$$

Just like what we did above, we assume $z \approx z_\phi$ near the sphere and thus we can omit the 2^{nd} order term:

$$r^2 + 2R\left(z_\phi\right)\left(z - z_\phi\right) = 0 \tag{3.54}$$

$$r^2 + \left[z - \left[z_\phi - R\left(z_\phi\right)\right]\right]^2 = R\left(z_\phi\right) \tag{3.55}$$

Therefore, we find out the radius of curvature of one point from the sphere M_ϕ with the same phase, where the centre of the sphere located at $\left(0, z_\phi - R\left(z_\phi\right)\right)$. As we check the boundary of $R(z)$, we found it reaches infinity around point z_0 :

$$\lim_{z \to z_0} R(z) = \infty \tag{3.56}$$

And for beam distance of infinity, we have:

$$\lim_{z \to \infty} R(z) = z - z_0 \tag{3.57}$$

Therefore, we can assume the gaussian beam to be a plane wave near the beam waist. However, for those parts of beam that are far away, it can only be considered as a spherical wave.

3.2.5 Gouy phase $\eta(z)$

Gouy phase is the phase shift as gaussian beam propagates away from the beam waist. We usually treat a gaussian beam as a plane wave, but we found that the beam is actually a spherical wave as it propagates far away. Therefore, there is a phase shift between the gaussian beam and a plane wave, and it is called the Gouy phase $\eta(z)$:

$$\eta(z) = \arctan \frac{z - z_0}{z_R} \tag{3.58}$$

At infinitely far away, it approaches to $\frac{\pi}{2}$. At Rayleigh range, it equals to $\frac{\pi}{4}$. In addition, Gouy phase will become larger with a higher order mode.

3.3 Hermite Gaussian mode

A gaussian beam have infinite mode at the same time. What we have calculated in the former section merely showed its basic mode, for we took the par-axial approximation of Helmholtz equation. When we consider the gaussian beam not as a simple plane wave, it will contain many higher order modes. If we decompose a coherent par-axial beam using the orthogonal set, we will get a group of higher order modes and they are called the Hermite Gaussian mode [17]. Here are two figures for the 00 mode (base mode) and 10 mode:

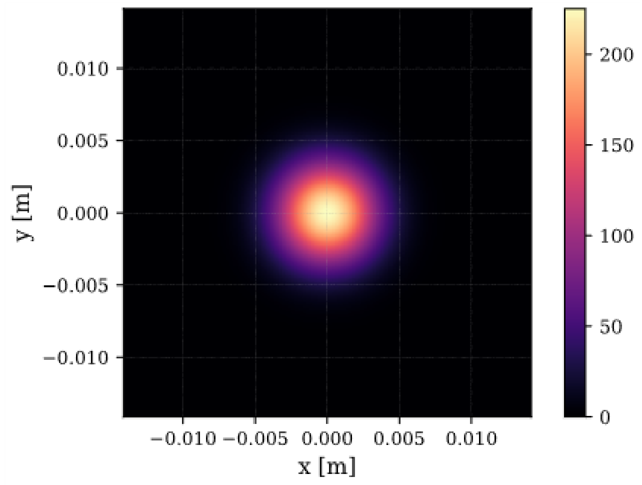

Figure 3.3.: Hermite Gaussian 00 mode

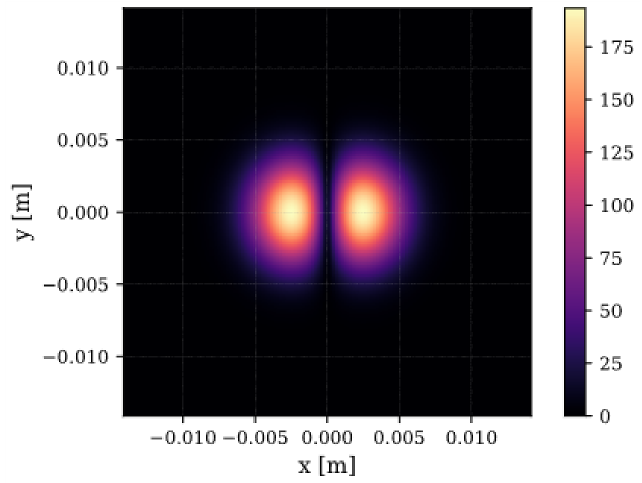

Figure 3.4.: Hermite Gaussian 10 mode

The general form of Hermite Gaussian mode is as follows:

$$U_{lm}(x, y, z) = U_l(x, z)U_m(y, z)e^{-ik(z-z_0)+i(l+m+1)\eta} \tag{3.59}$$

We see it is given by the product of a factor in x and a factor in y. The term $U_n(x, z)$

is the component of factor in x :

$$U_n(x,z) = \left(\frac{2}{\pi\omega^2}\right)^{\frac{1}{4}} \sqrt{\frac{1}{2^n n!}} H_n\left(\frac{\sqrt{2}x}{w}\right) e^{-\left(\frac{x}{w}\right)^2 - i\frac{kx^2}{2R}} \tag{3.60}$$

Where $H_n(x)$ is called Hermite polynomials and is as follows:

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2} \tag{3.61}$$

The Hermite Gaussian mode has the same beam parameter just as the one of basic mode has:

$$w^2(z) = w_0^2 \left[1 + \left(\frac{z - z_0}{z_R}\right)^2\right] \tag{3.62}$$

$$R(z) = (z - z_0)\left[1 + \left(\frac{z_R}{z - z_0}\right)^2\right] \tag{3.63}$$

$$\eta(z) = \arctan\frac{z - z_0}{z_R} \tag{3.64}$$

Consider the previous basic mode in notation of Hermite Gaussian mode:

$$\begin{aligned} U_{00}(x,y,z) &= U_0(x,z)U_0(y,z)e^{-ik(z-z_0)+i\eta} \\ &= \sqrt{\frac{2}{\pi\omega^2}} e^{-ik(z-z_0)+i\eta-(x^2+y^2)\left(\frac{1}{w^2}+\frac{ik}{2R}\right)} \end{aligned} \tag{3.65}$$

We can rewrite the general form of Hermite Gaussian mode in terms of basic mode as:

$$U_{lm}(x,y,z) = \sqrt{\frac{1}{2^l l! 2^m m!}} H_l\left(\frac{\sqrt{2}x}{w}\right) H_m\left(\frac{\sqrt{2}y}{w}\right) U_{00}(x,y,z) e^{i(l+m)\eta} \tag{3.66}$$

With the Hermite Gaussian mode explained, we can finally calculate the movement of the beam. It can be separated into two movement, translation and rotation.

For beam translation, we have the original coordinate as (x', y', z'), and the translated coordinate as (x, y, z). Assume a gaussian beam is translated along x-axis for a small translation δx. The basic mode of the beam can be represented as:

$$U_{00}(x,y,z) = U_{00}\left(x' - \delta x, y, z\right) \tag{3.67}$$

Like how we calculated beam parameter in the previous section, if we take the 1 st order Maclaurin expansion, we can have the beam translation for 00 mode and 10 mode as follows:

$$U_{00}(x,y,z)|_{z'=0} \approx U_{00}\left(x',y',0\right) + \frac{\delta x}{w_0} U_{10}\left(x',y',0\right) \tag{3.68}$$

$$U_{10}(x,y,z)|_{z'=0} \approx U_{10}\left(x',y',0\right) - \frac{\delta x}{w_0} U_{00}\left(x',y',0\right) \tag{3.69}$$

For beam rotation, we use similar coordinates as in translation and apply a small rotation $\delta\theta$. In form of matrix, the rotation is as follows:

$$\begin{pmatrix} x \\ z - z_0 \end{pmatrix} = \begin{pmatrix} \cos\delta\theta & \sin\delta\theta \\ -\sin\delta\theta & \cos\delta\theta \end{pmatrix} \begin{pmatrix} x' \\ z' - z_0 \end{pmatrix} \tag{3.70}$$

With the same assumption, the rotation of the beam with 00 mode and 10 mode can be represented as:

$$U_{00}(x,y,z)|_{z'=0} \approx U_{00}\left(x',y',0\right) + i\frac{\delta\theta}{\alpha_0} U_{10}\left(x',y',0\right) \tag{3.71}$$

$$U_{10}(x,y,z)|_{z'=0} \approx U_{10}\left(x',y',0\right) + i\frac{\delta\theta}{\alpha_0} U_{00}\left(x',y',0\right) \tag{3.72}$$

Where:

$$\alpha_0 = \frac{2}{k\omega_0} \tag{3.73}$$

3.4 Fabry-Perot Cavity

In this section, we will introduce an basic optical structure that is widely used in telecommunications, lasers and spectroscopy to control and measure the wavelengths of light [18]. It is applied in both arms of all interferometer-type gravitational wave detector. It is also the main structure of our setup for simulation, the Fabry-Perot cavity. A simple schematic figure of a Fabry-Perot cavity is as follows:

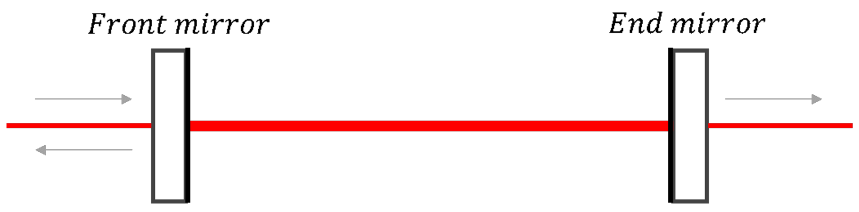

Figure 3.5.: Fabry-Perot Cavity

A Fabry-Perot cavity is a pair of mirrors in parallel position. Both mirrors have high reflectivity on the surface facing each other. For such structure, optical waves can pass through the optical cavity only when they are in resonance with it. Those resonant waves will travel back and forth through this cavity for a long time, in order to extend the optical path. The phenomenon here is called resonant enhancement. In general, the mirror on the laser source side is called front mirror and the other one is called end mirror. Usually, the end mirror has a higher reflection rate than the front mirror. Assume the input laser has the resonant frequency of the Fabry-Perot cavity. Its wave equation is given by:

$$E_{\text{in}} = E_0 e^{i\omega t} \tag{3.74}$$

We note the reflect coefficient for both mirrors as r_F, r_E and the transmit coefficient as t_F, t_E. The wave equation inside the cavity can be calculated by summing up the transmitted beam inside the cavity with all other beams that get reflected by the front mirror as follows:

$$\begin{aligned} E_{\text{cav}}(\phi) &= t_F E_{\text{in}} + r_E r_F t_F E_{\text{in}} e^{i\phi} + r_E^2 r_F^2 t_F E_{\text{in}} e^{2i\phi} + \cdots \\ &= \frac{t_F E_{\text{in}}}{1 - r_F r_E e^{i\phi}} \end{aligned} \tag{3.75}$$

Where ϕ is the phase shift for one back and forth process. It is proportional to the length of the cavity and is defined by:

$$\phi = \frac{2\omega L}{c} \tag{3.76}$$

The reflected wave equation can be calculated by taking the combination of the reflected component from the front mirror and the transmitted component from the cavity in the inverse direction:

$$\begin{aligned} E_r(\phi) &= r_F E_{\text{in}} + t_F E_{\text{cav}}(\phi) \\ &= \left(r_F - \frac{t_F^2 r_E e^{i\phi}}{1 - r_F r_E e^{i\phi}} \right) E_{\text{in}} \end{aligned} \tag{3.77}$$

And the transmitted wave equation is then:

$$\begin{aligned} E_t(\phi) &= t_E E_{\text{cav}}(\phi) e^{i\phi} \\ &= \frac{t_F t_E e^{i\phi}}{1 - r_F r_E e^{i\phi}} E_{\text{in}} \end{aligned} \tag{3.78}$$

From these two wave equations, we can calculate the reflection rate and transmission rate by the following division:

$$r_{\text{cav}}(\phi) = \frac{E_r}{E_{\text{in}}} = r_F - \frac{t_F^2 r_E e^{i\phi}}{1 - r_F r_E e^{i\phi}} \tag{3.79}$$

$$t_{\text{cav}}(\phi) = \frac{E_t}{E_{\text{in}}} = \frac{t_F t_E e^{i\phi}}{1 - r_F r_E e^{i\phi}} \tag{3.80}$$

If we take the square of the reflected wave equation and transmitted wave equation, we have the power of those two waves as follows:

$$\begin{aligned} P_r(\phi) &= \left|E_r(\phi)\right|^2 \\ &= \left|r_{\text{cav}}(\phi)\right|^2 P_{\text{in}} \\ &= \left[r_F - \frac{t_F^2 r_E e^{i\phi}}{1 - r_F r_E e^{i\phi}}\right]^* \left[r_F - \frac{t_F^2 r_E e^{i\phi}}{1 - r_F r_E e^{i\phi}}\right] P_{\text{in}} \\ &= \frac{\left[\left(t_F^2 + r_F^2\right) r_E - r_F\right]^2 + 4 r_E r_F \left(t_F^2 + r_F^2\right) \sin^2 \frac{\phi}{2}}{\left(1 - r_F r_E\right)^2 + 4 r_F r_E \sin^2 \frac{\phi}{2}} P_{\text{in}} \end{aligned} \tag{3.81}$$

$$\begin{aligned} P_t(\phi) &= \left|E_t(\phi)\right|^2 \\ &= \left|t_{\text{cav}}(\phi)\right|^2 P_{\text{in}} \\ &= \left[\frac{t_F t_E e^{i\phi}}{1 - r_F r_E e^{i\phi}}\right]^* \left[\frac{t_F t_E e^{i\phi}}{1 - r_F r_E e^{i\phi}}\right] P_{\text{in}} \\ &= \frac{\left(t_F t_E\right)^2}{\left(1 - r_F r_E\right)^2 + 4 r_F r_E \sin^2 \frac{\phi}{2}} P_{\text{in}} \end{aligned} \tag{3.82}$$

To simplify, we defined another parameter $F = \frac{4 r_F r_E}{(1 - r_F r_E)^2}$, the power can be rewrited into:

$$P_r(\phi) = \frac{\left[\left(t_F^2 + r_F^2\right) r_E - r_F\right]^2 + 4 r_E r_F \left(t_F^2 + r_F^2\right) \sin^2 \frac{\phi}{2}}{\left(1 - r_F r_E\right)^2 \left[1 + F \sin^2 \frac{\phi}{2}\right]} P_{\text{in}} \tag{3.83}$$

$$P_t(\phi) = \frac{\left(t_F t_E\right)^2}{\left(1 - r_F r_E\right)^2 \left[1 + F \sin^2 \frac{\phi}{2}\right]} P_{\text{in}} \tag{3.84}$$

The properties of a Fabry-Perot cavity depend on the parameters of the cavity. If we want to check out how the relative transmittance between the two mirrors affects the reflected, circulating, and transmitted fields in a clear view, we can build a simulation model in FINESSE. We will dive into this simulation system later in data generation section, but now we only took a glance at these different signals of Fabry-Perot cavity against a frequency offset. Figure 3.6 below shows the reflected signal, circulating signal and transmitted signal of a 4 km Fabry-Perot cavity. Note the circulating signal is detected on the side of front mirror:

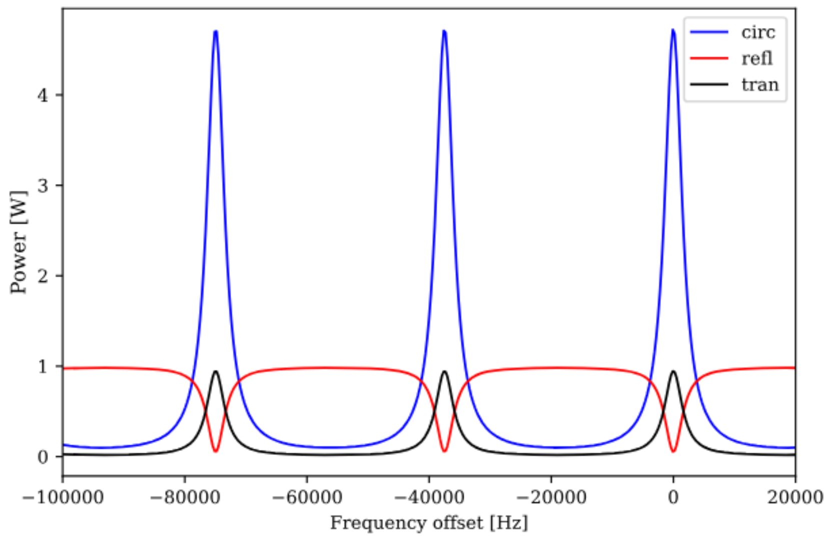

Figure 3.6.: Signals of Fabry-Perot Cavity

For a typical Fabry-Perot cavity, the resonant condition is:

$$\phi_c = 2n\pi \tag{3.85}$$

Under this frequency, the transmitted light and circulating light will reach their maximum power. Therefore, it is called the resonant state of cavity. We know this resonant condition depends on the angular frequency ω_c and the length L of the cavity as follows:

$$\phi = \frac{2L\omega}{c} \tag{3.86}$$

Based on the above formula, our resonant condition can be rewrite with regard of the angular frequency ω_c and the length L as follows:

$$L\omega_c = n\pi c \tag{3.87}$$

In figure 3.6, we can see several gaps between two resonant states. These gaps are called free spectrum range. There are two types of gaps depending on the variables. For a fixed laser frequency, the length gap of free spectrum range can be calculated by its frequency as follows:

$$L_{\text{FSR}} = \frac{c}{2f} \tag{3.88}$$

While for a fixed cavity length, the frequency gaps of free spectrum range can be calculated by its length using the same way:

$$f_{\text{FSR}} = \frac{c}{2L} \tag{3.89}$$

Usually, when we are talking free spectrum range, we mean the frequency gaps f_{FSR}.

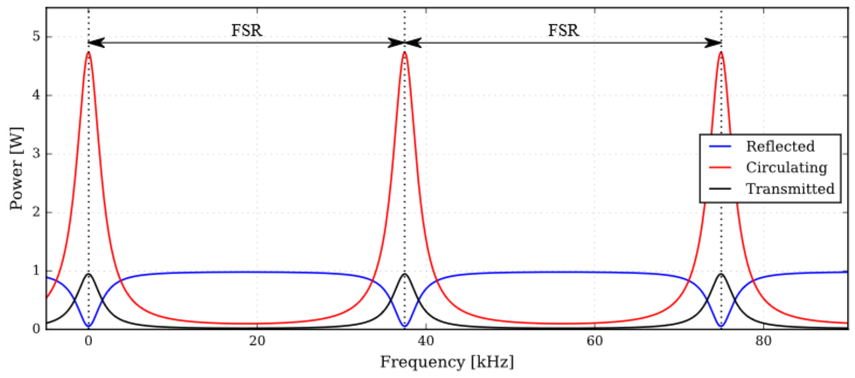

Figure 3.7.: Free Spectrum Range

In addition, it is obvious to see that the difference between every two resonant frequency and cavity length are proportional to each other. If we take several points on the resonant state of Fabry-Perot cavity and add a fitting line to it, we achieve the following figure:

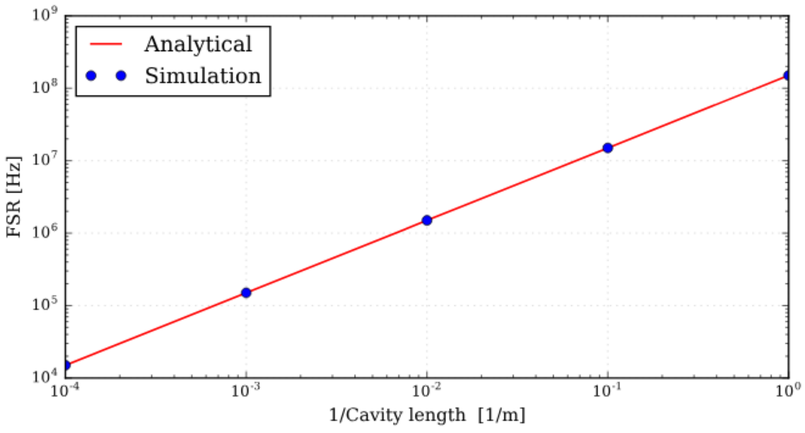

Figure 3.8.: Fitting line for FSR

We see that the signals from a Fabry-Perot cavity are periodical functions against frequency. The blue points are the intensity peaks of the resonant states. The red line is the fitting line. When we keep focusing on one of those peaks, we could calculate the full width at half maximum of that peak from the phase shift at half maximum power.

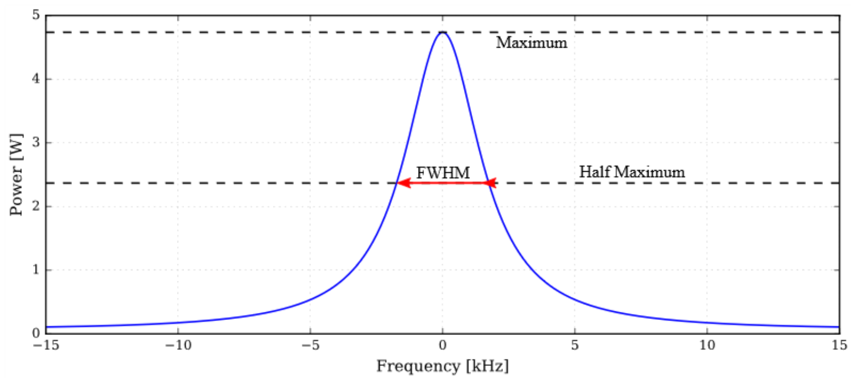

Figure 3.9.: Full Width Half Maximum

Simply bring in the power at resonant frequency and we can find:

$$P_t\left(\phi_{\text{half}}\right) = \frac{1}{2} P_t\left(\phi_c\right) \tag{3.90}$$

$$\frac{1}{1 + F \sin^2 \frac{\phi_{\text{half}}}{2}} P_{\text{in}} = \frac{1}{2} P_{\text{in}} \tag{3.91}$$

$$\sin \frac{\phi_{\text{half}}}{2} = \pm \frac{1}{\sqrt{F}} \tag{3.92}$$

$$\phi_{\text{half}} \sim \pm \frac{2}{\sqrt{F}} \tag{3.93}$$

Therefore, when we bring ϕ_{half} into frequency domain, we have the full width at half maximum in the form of free spectrum range:

$$f_{\text{FWHM}} = \frac{c\left(2\phi_{\text{half}}\right)}{4\pi L} = \frac{1 - r_F r_E}{\pi \sqrt{r_F r_E}} f_{\text{FSR}} \tag{3.94}$$

The frequency ratio between free spectrum range and full width of half maximum is called finesse, written in $\mathcal{F}$:

$$\mathcal{F} = \frac{f_{\text{FSR}}}{f_{\text{FWHM}}} = \frac{\pi \sqrt{r_F r_E}}{1 - r_F r_E} \tag{3.95}$$

3.5 PDH and WFS methods

The traditional method of mirror control is to first use the Pound-Drever-Hall (PDH) technique to lock the beam, and use the Wave-front sensing (WFS) signals to calculate the tilting angle of the mirror [19, 20]. With a modulated laser, we have a laser with side band as follows:

$$\begin{aligned} E &= E_0 e^{i(\omega t+\beta\sin\Omega t)} \\ &\approx E_0\left[J_0(\beta)e^{i\omega t}+J_1(\beta)e^{i(\omega+\Omega)t}-J_1(\beta)e^{i(\omega-\Omega)t}\right] \end{aligned} \tag{3.96}$$

Where β is modulation index, Ω is modulation frequency and J_n is Bessel function of the first kind. Here we only consider the first-order side band. The intensity of transmitted light is now:

$$\begin{aligned} P_t =& |E_t|^2 \\ =& P_0\left[|J_0(\beta)t(\omega)|^2+|J_1(\beta)t(\omega+\Omega)|^2+|J_1(\beta)t(\omega-\Omega)|^2\right] \\ &+2P_0J_0(\beta)J_1(\beta)\,\mathrm{Re}\left([t(\omega)t^*(\omega+\Omega)-t^*(\omega)t(\omega+\Omega)]\,e^{-i\Omega t}\right) \\ &+O(2\Omega) \end{aligned} \tag{3.97}$$

The first term is the DC component of the beam. If we omit the higher order term and treat the second term as our RF component, we have PDH signal as:

$$P^I_{\mathrm{demod}} = P_0J_0(\beta)J_1(\beta)\,\mathrm{Im}\left(t(\omega)t^*(\omega+\Omega)-t^*(\omega)t(\omega+\Omega)\right) \tag{3.98}$$

$$P^Q_{\mathrm{demod}} = P_0J_0(\beta)J_1(\beta)\,\mathrm{Re}\left(t(\omega)t^*(\omega+\Omega)-t^*(\omega)t(\omega+\Omega)\right) \tag{3.99}$$

The first signal is called the demodulated signal of in phase and the second one is called the demodulated signal of quadrature phase. We can stabilize the Gaussian beam by keeping the first signal maximized around the resonant frequency.

After the beam is stabilized, we start to consider the calculation for WFS signals [21, 22]. Assuming that we have an aligned Gaussian beam, the reflection rate of the 00 mode is already shown in previous section as:

$$r_{\mathrm{cav},00} = r_{\mathrm{cav}} = -r_F + \frac{t_F^2 r_E e^{i\phi}}{1-r_F r_E e^{i\phi}} \tag{3.100}$$

The 10 mode is different from 00 mode with a Gouy phase shift η_{FP} as:

$$r_{\mathrm{cav},10} = -r_F + \frac{t_F^2 r_E e^{i(\phi-2\eta_{\mathrm{FP}})}}{1-r_F r_E e^{i(\phi-2\eta_{\mathrm{FP}})}} \tag{3.101}$$

We can combine these reflection rate as a reflection matrix:

$$R^{\mathrm{align}}_{\mathrm{FP}} = \begin{pmatrix} r_{\mathrm{cav},00} & 0 \\ 0 & r_{\mathrm{cav},10} \end{pmatrix} \tag{3.102}$$

With a mis-alignment γ from the laser source, we have:

$$\begin{aligned}
R_{\text{FP}}^{\text{mis-align}} &= M^*(\gamma_r)\, R_{\text{FP}}^{\text{align}}\, M(\gamma) \\
&= \begin{pmatrix} 1 & \gamma_r^* \\ -\gamma_r & 1 \end{pmatrix} \begin{pmatrix} r_{\text{cav},00} & 0 \\ 0 & r_{\text{cav},10} \end{pmatrix} \begin{pmatrix} 1 & \gamma \\ -\gamma^* & 1 \end{pmatrix} \\
&= \begin{pmatrix} r_{\text{cav},00} - r_{\text{cav},10}\gamma_r^*\gamma^* & r_{\text{cav},00}\gamma + r_{\text{cav},10}\gamma_r^* \\ r_{\text{cav},00}\gamma_r - r_{\text{cav},10}\gamma^* & -r_{\text{cav},00}\gamma\gamma_r + r_{\text{cav},10} \end{pmatrix}
\end{aligned} \tag{3.103}$$

Omitting the second order term of the mis-alignment, we can simplify the matrix as:

$$R_{\text{FP}}^{\text{mis-align}} = \begin{pmatrix} r_{\text{cav},00} & r_{\text{cav},00}\gamma + r_{\text{cav},10}\gamma_r^* \\ r_{\text{cav},00}\gamma_r - r_{\text{cav},10}\gamma^* & r_{\text{cav},10} \end{pmatrix} \tag{3.104}$$

If we assume the laser is generated with only the base mode, we have our mis-aligned reflection rate as:

$$r_{\text{cav}}^{\text{mis}-\text{align}} = r_{\text{cav},00}U_{00} + \left(r_{\text{cav},00}\gamma_r - r_{\text{cav},10}\gamma^*\right)U_{10} \tag{3.105}$$

To simplify, we note r_c as the reflection rate of the gaussian beam, r_s as the reflection rate of the side band, we can rewrite the PDH signal of input phase as:

$$P_{\text{demod}}^{I} = iP_0 J_0(\beta) J_1(\beta) \left(r_c^* r_s - r_c r_s^*\right) \tag{3.106}$$

We can calculate the interference of the beam and its side band as follows:

$$\begin{aligned}
r_c^* r_s - r_c r_s^* = {} & U_{00}U_{00}^* \left(r_{c0}^* r_{s0} - r_{c0} r_{s0}^*\right) \\
& + U_{00}U_{10}^* \left[r_{c0}\left(r_{s0}\gamma_r^* + r_{s1}\gamma\right) - r_{s0}\left(r_{c0}\gamma_r^* + r_{c1}\gamma\right)\right] \\
& + U_{10}U_{00}^* \left[r_{10}^*\left(r_{c0}\gamma_r + r_{c1}\gamma^*\right) - r_{c0}^*\left(r_{s0}\gamma_r + r_{s1}\gamma^*\right)\right]
\end{aligned} \tag{3.107}$$

The first term is the PDH signal of the base mode, which equals to 0 and will be omitted when it is in resonant state. The last two terms are the interference of the 00 mode and 10 mode, which is the components of WFS signal. We note $U \equiv U_{00}U_{10}^*$ and rewrite the components into:

$$\begin{aligned}
W &= U\left[r_{c0}\left(r_{s0}\gamma_r^* + r_{s1}\gamma\right) - r_{s0}\left(r_{c0}\gamma_r^* + r_{c1}\gamma\right)\right] \\
&\quad + U^*\left[r_{10}^*\left(r_{c0}\gamma_r + r_{c1}\gamma^*\right) - r_{c0}^*\left(r_{s0}\gamma_r + r_{s1}\gamma^*\right)\right] \\
&= \left(r_{c0} rs_1 + r_{c1} r_{s0}\right)\left(U\gamma - U^*\gamma^*\right) \\
&= \left(r_{c0} rs_1 + r_{c1} r_{s0}\right)\left[\left(U - U^*\right)\frac{\delta x}{w_0} + \left(U + U^*\right) i\frac{\delta\theta}{\alpha_0}\right]
\end{aligned} \tag{3.108}$$

Bring back U and U^* and we have the WFS signal under the resonant frequency as:

$$\begin{aligned} P^I_{\mathrm{WFS}} &= iP_0 J_0(\beta) J_1(\beta) W \\ &= 2P_0 J_0(\beta) J_1(\beta) U_1^* U_0^* U_0 U_0 \left(r_{c0} r s_1 + r_{c1} r_{s0}\right) \left(\frac{\delta x}{w_0} \sin\eta - \frac{\delta\theta}{\alpha_0} \cos\eta\right) \end{aligned} \tag{3.109}$$

Chapter 4

Deep Learning

In this chapter, we will introduce some theories in computer science rather than physics. In this study, the main idea is using deep learning technology to build an artificial neural network that has the function of a control system. Though the title of this chapter is named by deep learning technology, we need to make a clear statement that deep learning is a sub-field of machine learning methods [23]. The following figure shows the relationship between deep learning, machine learning and artificial intelligence:

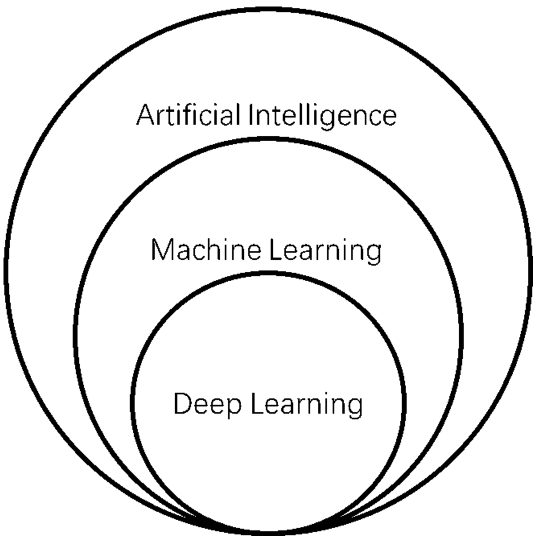

Figure 4.1.: Sub-fields of AI

Since there are many other methods under this topic, we will begin with a brief introduction on different approaches of machine learning. Then follows up with the details of those methods we used in this study: principal component analysis and two types of neural networks. After we finish all the structures of these methods, we will end up this chapter by introducing details during the learning process.

4.1 Machine Learning

Artificial beings with intelligence can be traced back to Greek mythology. As the development of computer hardware allows people to save massive amount of structured data, the idea of artificial intelligence starts to show up. Machine learning is the experimental branch of artificial intelligence, which means making machines to "learn" the way of solving a particular problem. Usually, the approaches of machine learning can be divided into three categories: supervised learning, unsupervised learning and reinforcement learning. They are categorized by the type of learning results and how these results change the learning models.

Supervised learning is a machine learning method for problems where there is a clear goal. It is used for solving complex problems that has massive data. Data sets of supervised learning consist of two parts, the training sets and the label sets (or sometimes called target sets). Traditional algorithms such as Bayesian statistics and decision tree are supervised learning. In this study, the artificial neural network we built belongs to this category.

Unsupervised learning is another method that handles massive data. The different of this type of learning from supervised one is that there are no label sets during learning period. A typical example for unsupervised learning is clustering. Later in this chapter, we will introduce an algorithm called principal component analysis (PCA), which we will see how the models transfer our raw data sets into something that contains less redundant information.

Reinforcement learning is another basic method that has different way of learning. This kind of learning emphasis on taking action to the whole environment rather than certain data sets. Unlike the other two methods, reinforcement does not have direct feedback through back-propagation to optimize the model. Instead, it seeks for a balance between the current information and new information from the environment. An example for this kind of learning would be the game theory. We would dive into this method for there is no usage of this method in this study.

By far, all three well used machine learning methods are introduced. There are also lots of different methods that can be roughly categorized into these three methods but have their unique characteristics. For this study, only the first two methods are used and they share the same algorithm while training.

4.2 Principal Component Analysis

Principal component analysis (PCA) method is a kind of statistical analysis that helps simplify the complexity of a data sets and remove redundant information. It is the most basic method among multidimensional data analysis methods without external criteria. Figure 4.2 is a simple example of analysing 2-dimensional data sets by PCA method:

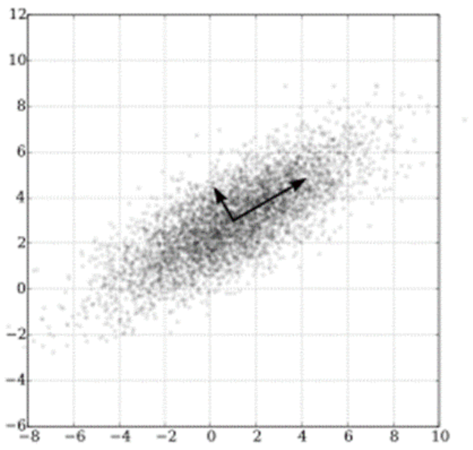

Figure 4.2.: Example of 2-dimensional PCA

It can be treated as a linear function that maps the original data to a new basis. The first step is to find the centre of all data in its original coordinates, choosing it as the new original points. Then it starts turning the axis to maximize the variance of all data. This axis is set to be the first coordinate, called the first principal component. The second step is to find the second axis whose correlation coefficient with the first component is 0 . This axis is then set to be the second coordinate.

For data sets with higher dimension, the second step will repeat until it all coordinates are found. The PCA method is often used when there are too many dimensions. If we set a proper number for the mapped coordinates, the redundant values within the original data sets will be merged into the other coordinates and make the new data set looks simpler. For the example in Figure 4.1, we could omit the second component if we need to further simplify the data set, for the values in the first coordinate already spread out and the difference between each other is clear enough.

4.3 Artificial Neural Network

The idea of using artificial neurons to help solving complex problems are proposed by Warren McCulloch and Walter Pitts [24]. The artificial neural network (ANN) is a mathematical model that is made up of multiple layers and nodes. ANN is inspired by the biological neural networks that constitute animal brains so that the aim of an ANN to learn and solve problems. Like a biological neural network, an ANN model connects its neurons with weights, and using nodes to represent neurons. A positive weight between two nodes reflects an excitational connection, while negative values mean inhibitory connections.

First, we will introduce the basic node within a ANN. A node saves the value of multiple weights and one bias. Each node will take multiple input through those weights and calculate the summation of these inputs with the bias. Figure 4.3 shows how a node adding up all its input:

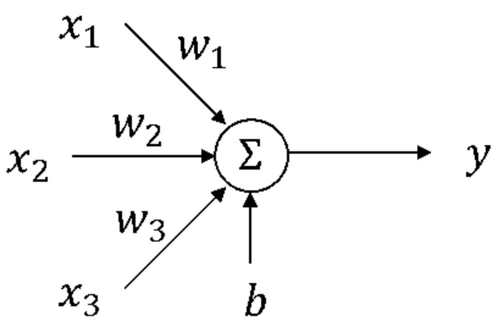

Figure 4.3.: Node

The summation a node simply using the following formula:

$$y = \sum_{i=1}^{n} w_i x_i + b \tag{4.1}$$

We can say this process act just like a linear combination. Therefore, sometimes ANN is also called linear neural network. Then, we combine these separated nodes to form a model. A model contains multiple layers, and each layer is made up of lots of nodes. A simple schematic figure of an ANN model is as follows:

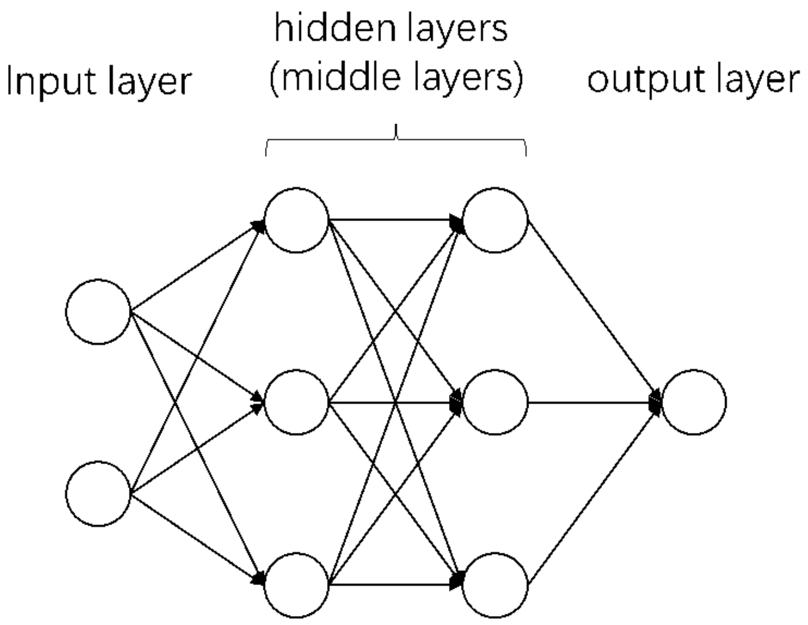

Figure 4.4.: Artificial Neural Network

In this figure, we can see that there are three types of layers and each layer has multiple nodes.

Input layer on the left-hand side layer is the input side of ANN. The function of this layer takes in the training data in pass them into the node. The input layer does not contain the summation process, but just passing one value to each node.

Hidden layer in this figure is the main part of ANN, or sometimes it is just called middle layer. The number of hidden layers may vary. It depends on the complexity of the data we want to handle. A larger amount of input value usually means there needs to be more hidden layers to analyse.

Output layer on the right-hand side returns the result of this ANN model. There are two types of output for deep learning model, classification and regression. For a classification model, the output layer contains multiple nodes whose number is equal to the number of labels. For example, a cat-dog classification model contains an output layer with two nodes. Each output node in such model means the probability of the labelled feature. For a regression model, the output layer contains only one node, and the output from this node returns a float. Using the same cat-dog as an example, it is a cat if the model returns a number closer to 1 , and it is a dog if the model returns a number closer to 0 .

In addition, after every node returns an output, there is a function called activation function. An activation function will control the amplitude of the output before it is passed to the next layer. We will introduce more about activation function in the last section of this chapter.

When training an ANN model, we need to set a constant called epoch which decide how many loops we will train this model. This is necessary because an ANN model may fall into an over-fitting trap. Then, the model will continually be running the following training loop until it reaches the epoch number we set. This figure shows how an ANN model are trained:

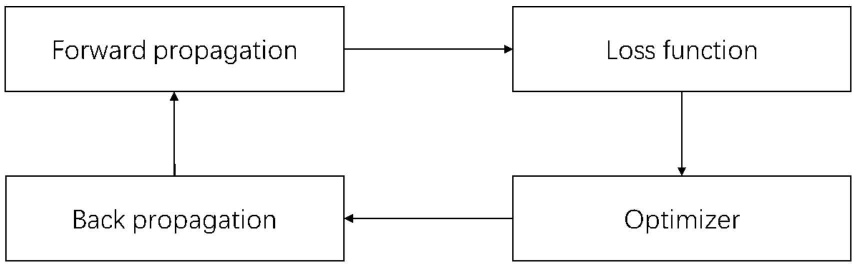

Figure 4.5.: Schematic of Training Loop

This loop contains four steps. First, it put the data set into the model and let the model calculate its output. Then, it compares the output with the label by a certain loss function and returns the difference between them. After that, the model will use a certain optimizer to find out how to change the weights and bias to minimize the loss. Finally, the difference from the loss function is applied to the model in the opposite direction, which is called back propagation. After the loop reaches the epoch number, we say the model is trained and can be saved for future experiment.

4.4 Convolutional Neural Network

Convolutional neural network (CNN) is the second type of model we used in this study and is proved to be an efficient algorithm with GPU-accelerating technology [25]. CNN model is a specific type of artificial neural network that is well known for its unique ability in analysing visual image. The earliest attempt of using CNN models is to classify the image of cats and dogs [26]. We know that in ANN model, nodes build up layers by a 1-dimension order. In CNN model, a layer is constructed in a 2dimension way. Therefore, there is a stronger relationship between the values who are neighbouring to each other. And the weights and bias were not saved in form of nodes, but in form of kernel (or filter). There are several features for CNN models. Some of them are the same as an ANN model, we just put our emphasis on its unique structures. They are convolutional layer, pooling layer, padding layer and channelling.

Convolutional layer calculates the convolution between the input and a kernel. Here is a figure shows how values are passed from one layer to another[27]:

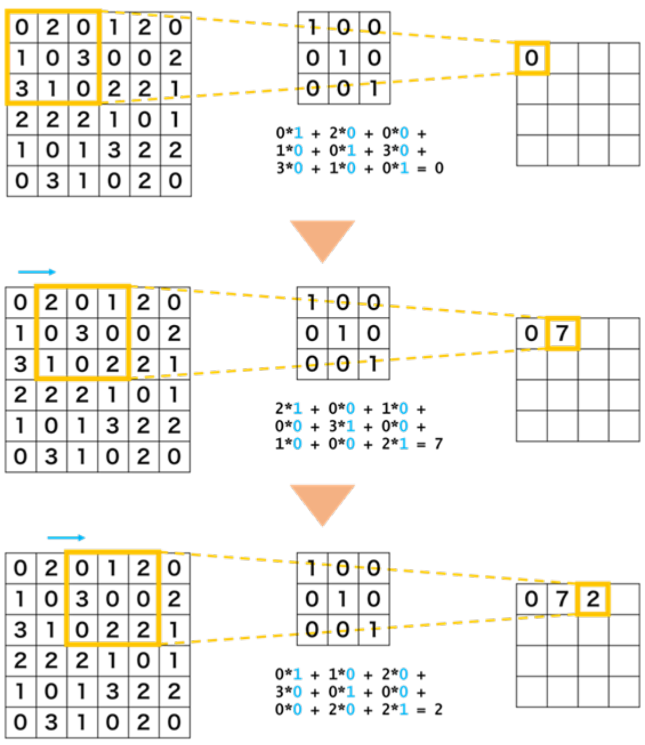

Figure 4.6.: Convolutional Layer

When data are passed through a convolutional layer, a kernel with the shape $n \times n$ will be applied on each area of data until it is fully covered. Its output would be a summation of those values with the kernel unit like what nodes do in ANN.

Pooling layer is used to reduce the number of values. We know convolutional layer in a CNN model will cause the reduction of value numbers. When there are too many values in a data set, pooling layer is a good choice. An example of pooling layer is as follows:

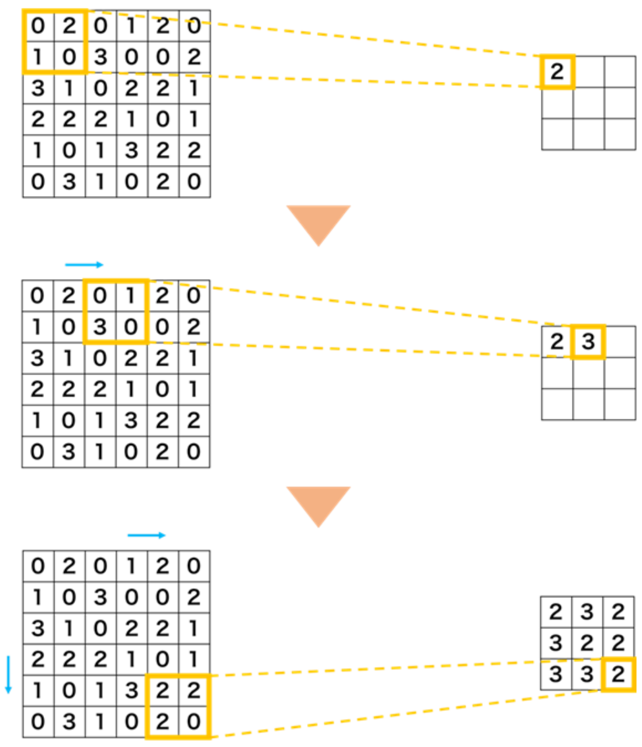

Figure 4.7.: Pooling Layer

Instead of applying a kernel, the pooling layer takes the maximum value of a certain area and return it as output. We can decide the size of the maximum area to control the speed of value reduction.

Padding layer increases the number of values by adding some space area on the outside of a visual image as the following figure:

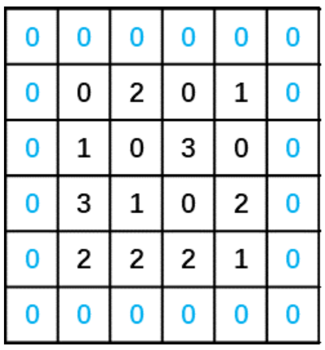

Figure 4.8.: Padding Layer

Though the padding layer adds a bunch of redundant value into the image, it is useful when there is a need in blurring the image. As long as the image is large enough, we can omit the negative effect brought by these zeroes.

For some small data set, we cannot build a model that is complex enough with only these layers, so there is another unique feature that is well used in CNN models. An image may have multiple channels. For example, an RGB image contains three channels where each channel saved the information of one colour. Not only for the difference in colours, any of the difference could be treated as different channels. Here is an example of increasing the channels of a cat image:

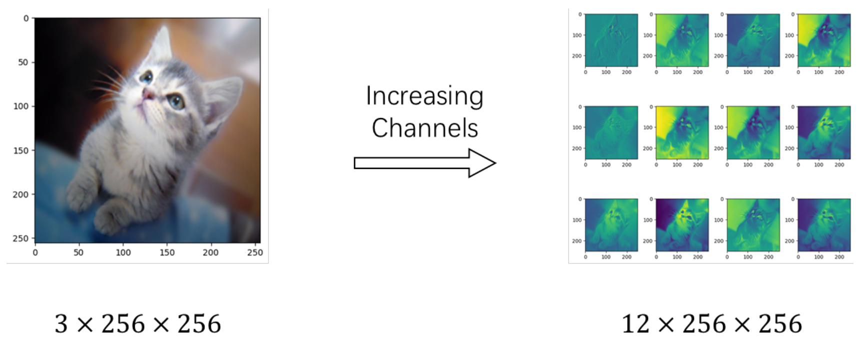

Figure 4.9.: Increasing channels

We can see from the figure that it creates several new channels that allows us to increase the complexity of our model. The image on the left saved its RGB information in three channels. After we apply four filters to each of its channel, the information of this image is transferred into 12 mono-colour channels. Which means the number of values is four times than before. This technique can be combined with other layers so that we can increase the number of channels and do convolutional calculation at the same time.

4.5 Loss Function

We have discussed about the loss function in the training loop. When a neural network is training, we need to evaluate the different between current output from the model and the true output it should be. The loss function compares the current output with the label and returns the difference between them. In addition, usually the batched loss function with more than one data is also called cost function. In this thesis, we will not put too much emphasis on this part and just call them loss functions. There are many different loss functions, each has a particular formula to calculate the difference. We will introduce three basic loss functions that are well used in deep learning.

4.5.1 L1Loss

L1Loss is a loss function for regression problems. It creates a criterion that measures the mean absolute error between each element in the output x and label y, it first calculated an un-reduced loss as follows:

$$L = l(x, y) = \{l_1, \dots, l_N\}^\top, \quad l_n = |x_n - y_n| \tag{4.2}$$

Here, N is the batch size of training. The batch size it a pre-defined constant that represent the number of samples that will be propagated through the network in the same time. Then the loss is calculated depend on the reduction we chose, either the mean value of L or the summation of L. The merit of L1Loss is that it has a fast convergent rate, which allows the gradient decent to find its direction in a higher accuracy. The disadvantage of L1Loss is also obvious. It has a high sensitivity on outlier.

4.5.2 MSELoss

MSELoss is a loss function for regression problems. It creates a criterion that measures the mean squared error between each element in the output x and label y. Therefore, it is also called M2Loss function. The formula of an MSELoss is as follows:

$$L = l(x, y) = \{l_1, \dots, l_N\}^\top, \quad l_n = (x_n - y_n)^2 \tag{4.3}$$

Like L1Loss, the loss is then calculated either using the mean value of L or the summation of L. The merit of MSELoss is that it is less affected by outliers. However, it is not continuous at original points, which means its not likely to return a valuable loss for small differences.

4.5.3 CrossEntropyLoss

CrossEntropyLoss [28] is a much complex loss function and is a loss function for classification problems. The formula for this loss function is:

$$L = l(x, y) = \{l_1, \dots, l_N\}^\top, \quad l_n = -\sum_{c=1}^{C} w_c \log \frac{e^{x,c}}{\sum_{c=1}^{C} e^{x_n,i}} y_{n,c} \tag{4.4}$$

The cross entropy comes from information theory. By definition, the cross entropy between two probability distributions p and q over the same underlying set of events measures the average number of bits needed to identify an event drawn from the set if a coding scheme used for the set is optimized for an estimated probability distribution q, rather than the true distribution p. Therefore, it is used to handle discrete losses and is useful in classification problems.

Of course, there are many other loss functions, such as SmoothL1Loss, CTCLoss or other type of entropy loss. In this study, we only use MSELoss for regression models and CrossEntropyLoss for classification models.

4.6 Optimizer

After we find the loss of training in one batch, we need to go back propagation. An optimizer in deep learning decides the calculation during bark propagation. Based on the definition of optimizers in deep learning, it will find the parameters θ of a neural network that significantly reduce a loss function $L(\theta)$, which typically includes a performance measure evaluated on the entire training set as well as additional regularization terms. Like the loss function, there are many optimizers to choose from. In this study, we used Adam optimizer. It is a well-used optimizer in recent deep learning studies. To understand Adam optimizer better, we have to start with other basic optimizers first. In the following paragraph, we will introduce from the original gradient descent optimizers. A short introduction of momentum optimizers will also be included. And we will end up with the development of optimizers with adaptive learning, which contains Adam.

4.6.1 Gradient Descent Optimizer

Gradient Descent [29] is the base optimizer of the first type. It is built on the gradient decent method in partial derivatives. It is an iteration method that is well used in unconstrained problems. The formula of gradient decent is as follows:

$$\theta_{t+1} = \theta_t - \alpha \cdot \nabla_\theta L(\theta) \tag{4.5}$$

Assume the parameter is θ, loss function is $L(\theta)$. The optimizer will update parameter θ by the gradient $\nabla_\theta L(\theta)$ with a learning rate α. Gradient decent is the simplest optimizer and it is later developed into the following new optimizers.

Batch Gradient Descent (BGD) is the optimizer with a batched loss function (cost function). The formula of batch gradient decent is as follows:

$$\theta_{t+1} = \theta_t - \alpha_t \cdot \frac{1}{n} \cdot \sum_{i=1}^{n} \nabla_\theta L_i \left(\theta, x^i, y^i\right) \tag{4.6}$$

Assume the batch size is n and the samples noted as $\left\{\left(x^1, y^1\right), \ldots, \left(x^n, y^n\right)\right\}$, and the model parameter is still θ. We have the gradient of sample $\left(x^i, y^i\right)$ as $\nabla_\theta L_i \left(\theta, x^i, y^i\right)$

with learning rate α_t. Since BGD needs to calculate the gradient for the whole batch while updating the parameters, this optimizer is slower than normal gradient decent. The merit of this optimizer is that it takes the mean gradient of each data point, so it is much likely to find the global solution.

Stochastic Gradient Descent (SGD) is another optimizer that it the opposite of the BGD optimizer above. It takes the result from a batched loss function and update the parameter with one randomly chose sample. The formula is as follows:

$$\theta_{t+1} = \theta_t - \alpha \cdot \nabla_\theta L_i \left(\theta, x^i, y^i\right) \tag{4.7}$$

Only one gradient is calculated by SGD among the whole batch. Therefore, the training speed is a lot faster than the previous optimizers. The disadvantage of SGD is also obvious. There will be a fluctuation in SGD training, which causes a high possibility for the model to jump between local optimum.

Mini-Batch Gradient Descent (MBGD) is the last optimizer we will introduce based on gradient descent. It is an optimizer that takes a balance between BGD and SGD. A MBGD formula looks like:

$$\theta_{t+1} = \theta_t - \alpha \cdot \frac{1}{m} \cdot \sum_{i=x}^{i=x+m-1} \nabla_\theta L_i \left(\theta, x^i, y^i\right) \tag{4.8}$$

For a batched data sets with n samples. It chooses a sub-batch with size m(m < n) which is the quite like a combination of BGD and SGD. It takes a balance between the training time and the accuracy. It avoided the fluctuation part that is well seen in SGD and thus will give a global optimum like BGD.

4.6.2 Momentum Optimizer

Momentum is the basic idea of all second type optimizers [30]. When updating the parameter, it will take the previous update into consideration as follows:

$$\begin{aligned} m_{t+1} &= \mu \cdot m_t + \alpha \cdot \nabla_\theta L(\theta) \\ \theta_{t+1} &= \theta_t - m_{t+1} \end{aligned} \tag{4.9}$$

The idea of momentum optimizer is that, to some extent, it remains the previous parameter θ, and combine the previous parameter with some fine adjustment. Therefore, when there is a change in direction of gradient, the momentum can lower the converge speed and prevent the fluctuation. For the same direction, the momentum optimizer will accelerate the converge speed of the model. Based on this idea, the following optimizer is developed.

Nesterov Accelerated Gradient (NAG) is a typical momentum optimizer. It fully remains the previous gradient and doing another adjustment after that. Its formula is as follows:

$$\begin{aligned} m_{t+1} &= \mu \cdot m_t + \alpha \cdot \nabla_\theta L\left(\theta - \mu \cdot m_t\right) \\ \theta_{t+1} &= \theta_t - m_{t+1} \end{aligned} \tag{4.10}$$

The change in loss function is called Nesterov term. Nesterov term allows the optimizer to adjust the current gradient after a large parameter update. Note that in PyTorch, there is no simple SGD optimizer, the optimizer called SGD is actually using momentum optimizer.

4.6.3 Optimizer with Adaptive Learning Rate

Adaptive Learning Rate leads to the third type of optimizers. We know that learning rate is usually a pre-defined constant that controls the balance between learning speed and fluctuation. An optimizer with adaptive learning rate will adjust the learning rate during the training period and help the model train faster.

Adaptive Gradient (AdaGrad) [31] is the first and simplest optimizer with adaptive learning rate. The optimizer is constituted by the following three steps:

$$\begin{aligned} r_{t+1} &= r_t + [\nabla_\theta L(\theta)]^2 \\ \Delta\theta &= \frac{\alpha}{\sqrt{r_{t+1} + \epsilon}} \cdot \nabla_\theta L(\theta) \\ \theta_{t+1} &= \theta_t - \Delta\theta \end{aligned} \tag{4.11}$$

We can see that r and ϵ here are some new parameters, r represents the gradient accumulation variable and $\epsilon \approx 10^{-8}$ is an extremely small constant. When updating parameter θ, the term $\frac{1}{\sqrt{r_{t+1}+\epsilon}}$ becomes a regularizing term. For a small r, the regularizing term is big so that it may enlarge the gradient, and vice versa. AdaGrad has all the merit as an optimizer with adaptive learning rate, but there are still many disadvantages. The learning rate α need to be set carefully or the regularizing term would be too sensitive to learn. One more problem is that when the updating part gets smaller and smaller, sometimes the model stopped learning before it reaches the global optimum. Therefore, many other improved optimizers have been designed after AdaGrad.

Adadelta [32] is an optimizer improved from AdaGrad. Adadelta changes the way of updating parameters. Gradient accumulation variable r is updated by a certain rate v. The formula is as follows:

$$\begin{aligned} r_{t+1} &= v \cdot r_{t-1} + (1-v) \cdot [\nabla_\theta L(\theta)]^2 \\ \Delta\theta_t &= \frac{1}{\sqrt{r_{t+1}+\epsilon}} \cdot \sum_{i=1}^{t-1} \Delta\theta_t \\ \theta_{t+1} &= \theta_t - \Delta\theta_t \end{aligned} \tag{4.12}$$

With an iteration on $\Delta\theta$, the optimizer no longer depends on the learning rate. By this method, the model keep training until it reaches a local optimum with a small fluctuation.

Root Mean Squared Propagation (RMSprop) is another optimizer improved from AdaGrad. The update of gradient accumulation variable r has been changed from squared summation into exponential average with weights as follows:

$$\begin{aligned} r_{t+1} &= \rho \cdot r_t + (1-\rho) \cdot [\nabla_\theta L(\theta)]^2 \\ \Delta\theta &= \frac{\alpha}{\sqrt{r_{t+1}+\epsilon}} \cdot \nabla_\theta L(\theta) \\ \theta_{t+1} &= \theta_t - \Delta\theta \end{aligned} \tag{4.13}$$

Using RMSprop optimizer still needs a learning rate α, but it also introduced a new parameter called decay rate $\rho \approx 0.9$. This method provided us another approach in reducing the updating speed of gradient accumulation variable r.

Adaptive Moment Estimation (Adam) [33] is the last optimizer we will introduce in this section. It is also the optimizer we use in this study. It is like a combination of Adadelta and RMSprop. In Adam optimizer, the momentum is introduced and take effects on the gradient update. The formula of Adam is given by the following 3 steps:

$$\begin{aligned} m_t &= \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot \nabla_\theta L(\theta) \\ v_t &= \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot [\nabla_\theta L(\theta)]^2 \\ \theta_{t+1} &= \theta_t - \frac{\alpha}{\epsilon + \frac{v_t}{1-\beta_2^t}} \frac{m_t}{1-\beta_1^t} \end{aligned} \tag{4.14}$$

Clearly, it is the most complex optimizer by far. The parameter is $\beta_1 = 0.9$ and $\beta_2 = 0.999$ by default. m_t and n_t are the first and second order estimation on momentum matrix. The feature of Adam optimizer has a stable learning rate after a few iterations, which also stabilize the parameter θ. It is automatically suitable for most problem with large data sets and complex data structure. And that is why we use Adam in our study.

4.7 Activation Function

An activation function is a function that is used right after each layer. By definition, an activation function decides the output of that node. The simplest example for activation functions is a digital network that can be either 1 of 0 , depending on input. Using different activation function will slightly affect the accuracy of the neural network. Here we will introduce four activation functions. The first one is the simple ReLU function, and the other three are PReLU, ELU and SELU functions which are the improved variants of ReLU. There are dozens of other activation functions in PyTorch, like sigmoid function and tanh function. We will not put too much emphasis on them. We simply take a look at the functions mentioned above as examples for understanding how activation functions may look like and why there are so many activation functions developed after ReLU function.

4.7.1 Rectified Linear Unit (ReLU) function

ReLU is the simplest activation function. It is set to be the default activation function for many neural network samples because it makes people easier to understand the importance of an activation function in a deep learning model. Basically, it will drop the negative value and only return the positive values as follows:

$$\mathrm{ReLU}(x) = (x)^+ = \max(0, x) \tag{4.15}$$

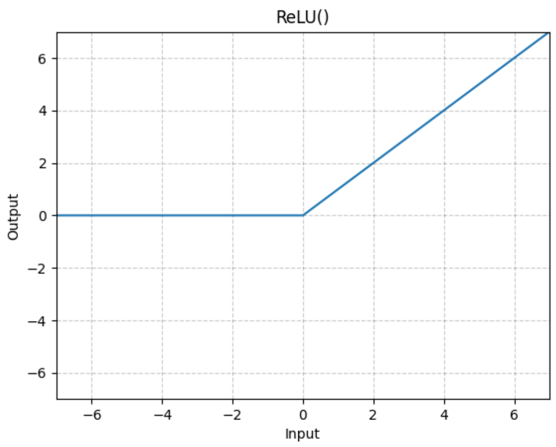

Figure 4.10.: ReLU function

4.7.2 Parametric Rectified Linear Unit (PReLU) function

PReLU is an activation function developed by ReLU function. It changes the negative half of ReLU function and allows a negative signal as follows:

$$\mathrm{PReLU}(x) = \max(0, x) + a * \min(0, x) \tag{4.16}$$

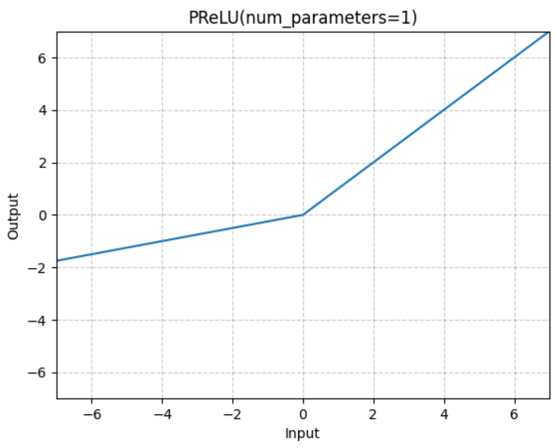

Figure 4.11.: PReLU function

PReLU functions add a learning parameter a (default: 0.25) that controls the negative half of the activation function. To some extent, it raises the fitting probability for each learning loop and only has low possibility causing over-fitting.

4.7.3 Exponential Linear Unit (ELU) function

ELU is another activation function that returns negative values, its definition is as follows:

$$\mathrm{ELU}(x) = \max(0, x) + \alpha * \min\left(0, e^{x} - 1\right) \tag{4.17}$$

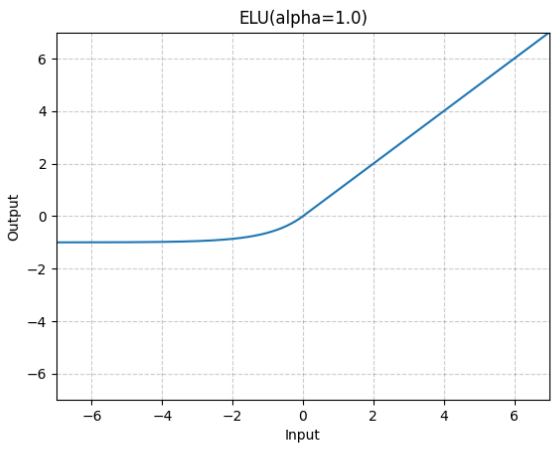

Figure 4.12.: ELU function

From the figure we can see that there is a limit in negative returns, which has the same merit as PReLU function and can prevent over-fitting with any parameter α (default: 1). Therefore, it is a balanced choice for most situation.

4.7.4 Scaled Exponential Linear Unit (SELU) function

SELU is an activation functions that induce self-normalizing properties. The function is developed by ELU function. The formula and figure are as follows:

$$\begin{aligned} \mathrm{SELU}(x) = &\ \text{scale } *(\max(0, x)+\min(0, \alpha *(e^x-1))) \\ &\alpha \approx 1.6733, \quad \text{scale } \approx 1.0507 \end{aligned} \tag{4.18}$$

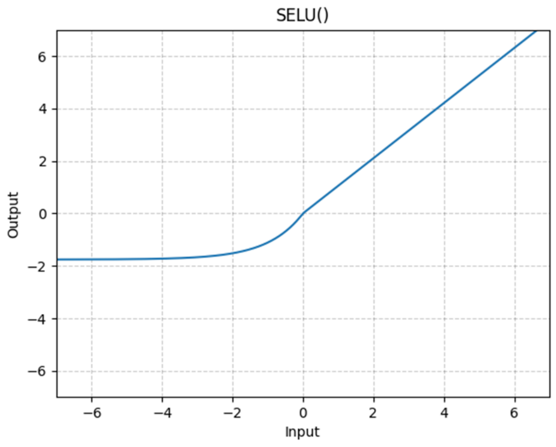

Figure 4.13.: SELU function

SELU function controls the total output with a scalar. And the scalar gives SELU function the property of internal normalization which helps adjust the mean and average of the layers. Comparing SELU with the above activation functions and we can see that all of these functions have nearly the same shape. It is one of the improved version of ReLU function and contains all the merit of ReLU function. In this study, we are using SELU function for our ANN models.

4.7.5 Sigmoid function

Sigmoid is an activation function that is out of the ReLU family and has its own characteristics. The formula and figure of Sigmoid are as follows:

$$\mathrm{Sigmoid}\,(x) = \delta\,(x) = \frac{1}{1+e^{-x}} \tag{4.19}$$

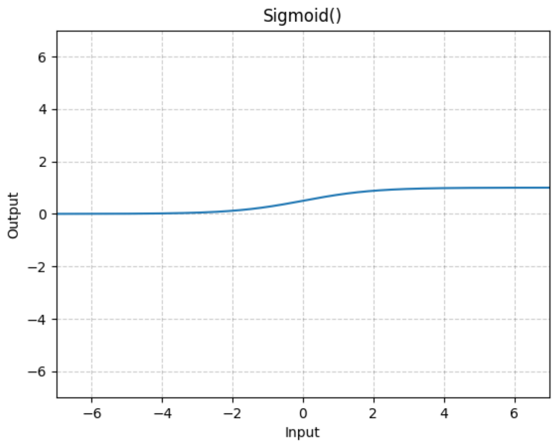

Figure 4.14.: Sigmoid function

Sigmoid function is a special activation function. It always give positive feedbacks to the model. From the formula we know that Sigmoid function is a non-linear function, which means the output will also be a non-linear function of the weighted sum of inputs. The merit of such function is that it provide a steady learning speed, while the disadvantage of such function is that it is much likely to cause an over-fitting problem.

4.7.6 Tanh function

Tanh function is another activation function out of the ReLU family. It is named by the hyperbolic tangent function tanh. Its formula and figure are as follows:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{4.20}$$

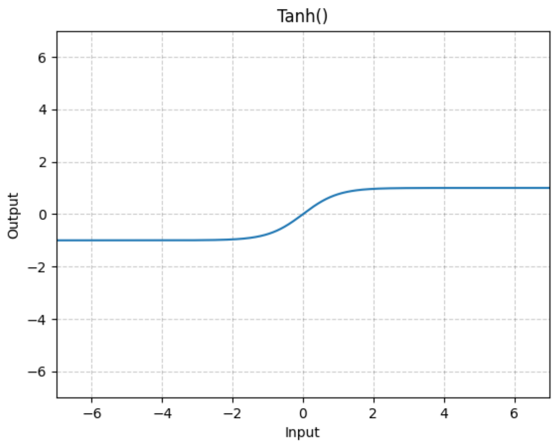

Figure 4.15.: Tanh function

By using Tanh function, we will achieve a stead learning speed like Sigmoid function. Meanwhile, comparing the feedbacks on the negative input with Sigmoid function, such a negative feedback prevent the model from falling into the over-fitting problem to fast like the previous models.

Among all these functions, there is no particular activation function that is the correct choice for all models. SELU is used in this study is because it is the most developed function and our model using this function returns a better accuracy after a few test runs.

Chapter 5

Experiment

Now that all the foundations are built, we finally move into our own study. Our study can be divided in three main parts: generating data, data analysis and applying deep learning. In this chapter, we will first take a review on the method through the whole study. Then, we will show the environment set for this study. In the end of this chapter, we will dive into these three parts for more details on how they work.

5.1 Methods of study

Our study can be divided into the following steps:

1. Launch finesse2 with PyKat to set up the optical simulation and generate a simple raw data set with 100 samples, at 4×4 resolution.
2. Use scikit-learn to apply PCA on the raw data and transfer it into matrix form for deep learning.
3. Build four type of deep learning models for comparison with PyTorch. Put the transformed data set into these models to make sure the codes are successfully running.
4. Generate several raw data sets with different precision, each with 10000 samples. Apply PCA on them and put them into the first deep learning model. Train and check how much precision we can get.
5. Generate massive rat data sets with different mirror type, each with 10000 samples. Apply PCA on them and put them into all four deep learning models. Train and check which model gives us the best result.

5.2 Environment of study

For this study, we use the following system environment:

- OS: Microsoft Windows 11 pro
- CPU: AMD Ryzen 5 3600 6-Core Processor 3.59Hz
- GPU: NVIDIA GeForce RTX 2060
- RAM: 32.0 GB

In order to reproduce the results of this study, the following coding environment is recommended:

- Python: 3.9.0
- Notebook: 6.5.2
- PyKat: 1.2.81
- scikit-learn: 1.2.1
- torch: 1.13.1

5.3 Generating data

What we need from the simulation is beam detector signals under different angles with different maps. By far, we do not know how complex a neural network can learn. We assume the best result for a trained model is that it will be able to return a correct tilting angle regardless of roughness for the mirror. We also expect the least result is that it only returns the correct angle when the system contains only perfect flat mirror. Therefore, we make four data sets with different kinds of mirror maps . The first data set contains no mirror map that represent the perfect flat mirror, noted as ”nomap”. The second data set contains only one mirror map with a pre-defined random roughness, noted as ”map”. The third data set contains a group of random mirror map generated by random numbers, noted as ”random”. The last data set contains one mirror map but with is mis-centered laser input, noted as ”miscenter”. In this section, we will introduce how we use finesse2 to run an optical simulation of Fabry-Perot cavity and generate the raw data sets.

Finesse2 is an interferometer simulation program that is originally written under C language. All of its code can be run under a script-like file called kat. We use a wrapping package PyKat to run finesse2 in Python language. Here is an example on how to run a Fabry-Perot cavity simulation with Python:

```
# Sample Code for Fabry-Perot Cavity
from pykat import finesse
```

```
kat1=finesse.kat() # initialising Finesse
kat1.verbose = False

basecode = """
## The laser ##
l laser 1 0 n0   # Laser (Power=1W, wavelength offset=0)
s s1 1 n0 n1    # Space (Length=1m)

## The beam splitter ##
s sbs1 0 n1 n2
bs bs 0.1 0.9 0 45 n2 dump n3 n4  # Beam splitter (Angle=45)
s sbs2 0 n3 nc1

## The cavity ##
m m1 0.7 0.3 0 nc1 nc2  # Mirror  (R=0.7, T=0.3, phi=0)
s sL 4000 nc2 nc3       # Space (Length = 4 km)
m m2 0.8 0.2 0 nc3 nc4  # Mirror  (R=0.8, T=0.2, phi=0)

## Detectors ##
pd refl nc1          # Reflected field
pd circ nc2          # Circulating field
pd tran nc4          # Transmitted field

## Simulation instructions ##
xaxis m1 phi lin -450 90 2000   # Tuning of input mirror
yaxis abs
"""

kat1.parse(basecode) # Parsing the FINESSE-code
out1 = kat1.run() # Running the FINESSE-simulation
```

First, an empty kat file are created named "kat1". Codes with red highlights are the scripts for kat file. It is consist of optical equipment, spaces and a variable. Optical equipment are added by its type, name, parameters and connections. Spaces connects with optical equipment are defined by its length and connections. These two components build up the Fabry-Perot cavity. The variable decides the output of this simulation. It is usually defined by moving or tilting a certain parts within the set-up. After the scripts are written, we pass them into the kat file we created and the simulation is able to run.

Now we understand how a simulation is generated through finesse2, the next step is to build up the set-up for a Fabry-Perot cavity. The cavity we train should be a part of gravitational wave detector, so it is actually a part of a Michelson interferometer. Thus, we use beam splitter to divide the laser into two direction, but dump the signals on the vertical arm. The whole set-up is as follows:

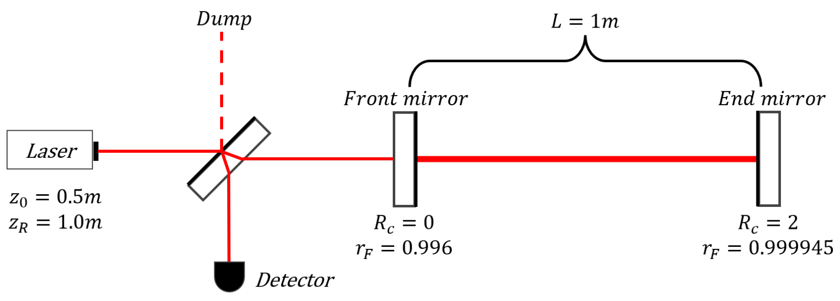

Figure 5.1.: Simulation Set-up

Note that we have modulated laser and curved mirror with our study. We also need to add the optimization for Gouy phase to the mirror. In Python, these attributes can be modified easily. After we set up a Fabry-Perot cavity with beam splitter like the codes above, we only need to parse some new lines of scripts into the basecode kat file as follows:

```
# Adding attributes
code = """
## Modulation ##
const fsb1 15M # Frequency modulation
const mfsb1 -15M

## Attributes ##
attr ITM Rc 0 # Adding radius of curvature to a mirror
attr ETM Rc 2
attr gouy_phase_tuner g 90 # Apply optimized Gouy phase by WFS
"""

kat1.parse(code)
```

Since the basic version of our Fabry-Perot cavity is built, there are only two more steps to go. To add mirror maps on the front mirror, and to replace the variable with the rotation of the end mirror. A mirror map used in finesse is a text file which contains the information of this map and a table of reflectivity. Usually the table consists of measured values from a randomly generated continuous functions. A simple example of a map file is down below, where the beam can only be transmitted from the center of this mirror:

```
% Surface map
% Name: map1
% Type: absorption both
% Size: 6 6
% Optical center (x,y): 3.5 3.5
% Step size (x,y): 1e-003 1e-003
% Scaling: 1
1       1       1       1       1       1
1       1       1       1       1       1
1       1       0       0       1       1
1       1       0       0       1       1
1       1       1       1       1       1
1       1       1       1       1       1
```

In our study, we need a much more precise mirror map instead of this 6×6 example. We generated 100 random maps with the following formula:

$$F(x) = r_1 * \cos(n_x \pi x + r_2 \pi) * \cos(n_y \pi y + r_3 \pi) \tag{5.1}$$

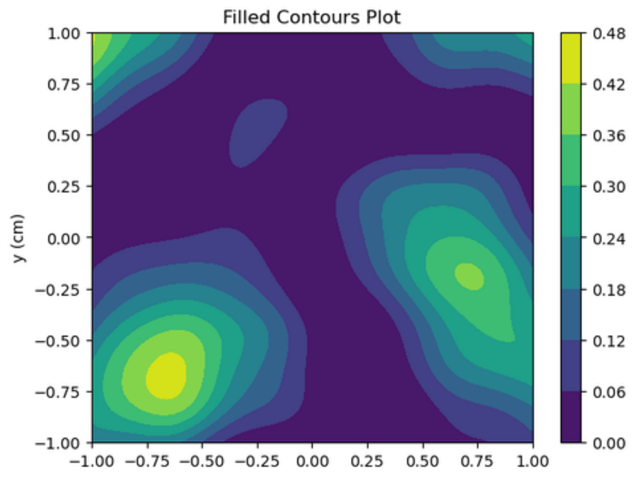

Figure 5.2.: Random map

After we have the random maps generated, we take 101×101 samples from this continuous maps to create the discrete map which fits the simulation in finesse2:

```
# Generating random maps
for i in tqdm(range(map_number)):
    map = [[0 for _ in range(101)] for _ in range(101)]
    for x in range(101):
        for y in range(101):
            map[x][y] = absorption(x/100, y/100, seed)
                        ** 2
            map_arr = np.array(map)
            np.savetxt(f'./mirror_map/mirror_map_{i}.txt',
                        map_arr)
            textfile.insert(f'mirror_map/mirror_map_{i}.txt',
                        '''
                        % Surface map\n
                        % Name: sub\n
                        % Type:absorption both\n
                        % Size: 101 101\n
                        % Optical center (x,y): 51 51\n
                        % Step size (x,y): 1e-004 1e-004\n
                        % Scaling: 1e-1\n
                        ''', line = 0)
```

In figure 1.1, we saw the signal of a Gaussian beam. When such a map is applied to the beam, the center of the beam will become hard to analysis, an sample plot will be the following:

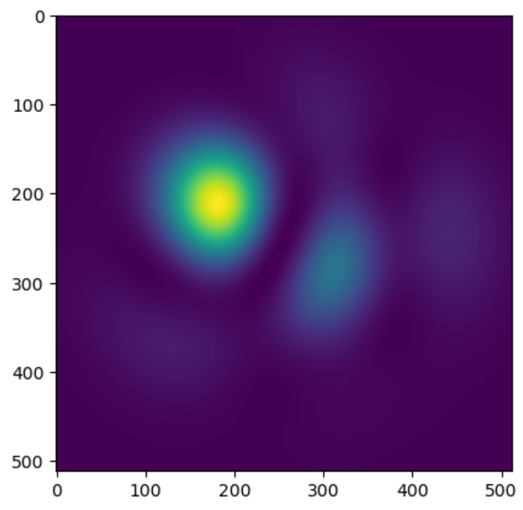

Figure 5.3.: Gaussian beam with mirror map

It is time to add the new variable into the simulation. Because we want to make analysis on 4×4 intensity array. In simulation, we can simply take the signals from a beam detector and set the sample rate at 4×4. We replaced those photon detectors in previous codes with a beam detector. We also need to add the rotations of the mirror by adding attributes to the script. We prepared a loop to change the tilting angle and insert them by Python f-string and the variable ”tilt” down below is the tilting angle calculated outside the script. Parsing those maps, variable, and new attributes into the simulation and we have the additional codes as follows:

```
# Adding maps, variable and new arributes
code = """
## Mirror map ##
map ITM './mirror_map/mirror_map_{data_count}.txt'
knm ITM sub_map
conf ITM save_knm_binary 1
conf ITM interpolation_method 2
conf ITM integration_method 1

## Attributes ##
attr {variable_mirror} xbeta {tilt}
attr gouy_phase_tuner g 90 #optimized

## Simulation instructions ##
beam ccd REFL
xaxis ccd x lin -2 2 {ccd_size-1}
x2axis ccd y lin -2 2 {ccd_size-1}
"""

kat1.parse(code)
```

Using the above Python codes, we are finally able to create the four data sets with the following parameters:

Table 5.1.: Parameters of Data Sets

Mirror	ETM rotation			
Map-type	no-map	map	random	mis-center
Map-number	0	1	100	100
Step number	10000	10000	100	100
CCD size	4 * 4			
Total data number	10000			

Each data sets contains 10000 samples like the following plot:

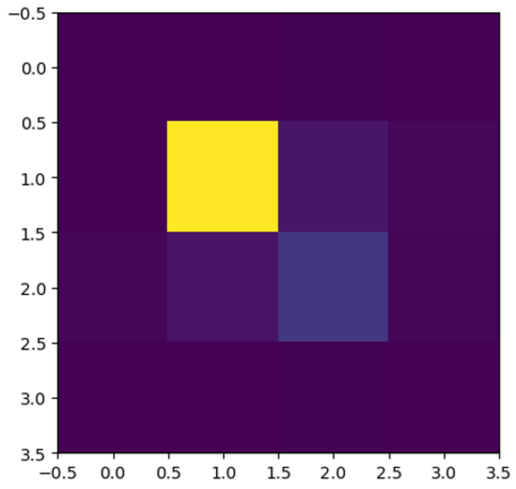

Figure 5.4.: Sample Raw Data

By far, we have explained how to generate four groups of data sets by finesse2 and some detailed parameters of them. The precision of these data sets are defined by the bound ranges and the step sizes. These two parameters will vary through the main steps. Therefore, we actually creates more than four data sets. We only need to keep in mind that no matter the precision, the rest of parameters are always the same as is shown above.

5.4 Data Analysis

Now we have four different type of data sets, each with 10000 intensity samples and the resolution is 4×4. What we need to do next is to apply principal component analysis (PCA) to these data for removing the redundant information. After the raw data sets get through a PCA model, we also need transform these data sets into array-like form so that the deep learning model can read them.

We will get into the details of how we apply PCA to our data sets first. In chapter IV, we have already introduced the basic idea of PCA and showed a 2-dimensional example. We know that PCA is an unsupervised learning model that can be trained with input data sets. Our raw data sets contains 16 intensity values, so we can treat as we have 16 dimensions. For a PCA model, if it is trained by a m-dimensional data sets, the output can have at most n-dimension for any $n < m$. However, the reduction of dimensions is not necessary for all PCA models. In our case, 16 eigenvalues is already a small number for a deep learning model. Thus, we do not want to reduce the dimension further. What we do is just to put one of the raw data sets into an empty PCA model, telling it to return the same dimension with the raw data set. By doing this, we will get a sorted data sets that put the principal components at the front and leave the less relevant components to behind. Here is a schematic for how we use PCA to handle our data sets:

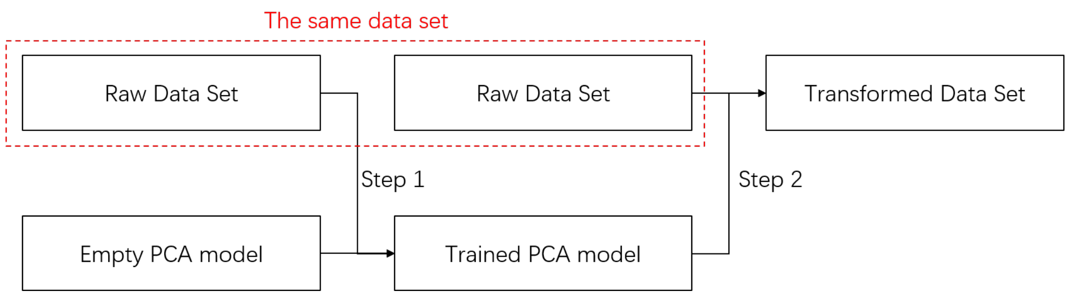

Figure 5.5.: Schematic of PCA

For this step, we use a package called scikit-learn. Scikit-learn is a tool package for predictive data analysis that is built on other basic python packages such as numpy, scipy, and matplotlib. The scikit-learn package [34] helps us apply PCA method to our raw data sets. We import this package and set the number of output be equal to the input intensity number as follows:

```
# Training and applying PCA model
from sklearn.decomposition import PCA

pca = PCA(copy=True, n_components=ccd_size**2, whiten=False)
pca.fit(ipt) # Training PCA model with a raw data set
print(pca.components_.shape)
ipt_transformed = pca.transform(ipt) # Apply PCA
```

We first defined an empty PCA model with the correct component number. By fitting our raw data set to the empty PCA model, it is fully trained. We then simply input the same data set once again into the model and the result will be the transformed data set.

When this PCA model is trained by a raw data set with 16 intensity values, we get 16 new basis. These basis are ordered by their importance to the data set. A sample of these basis is as follows:

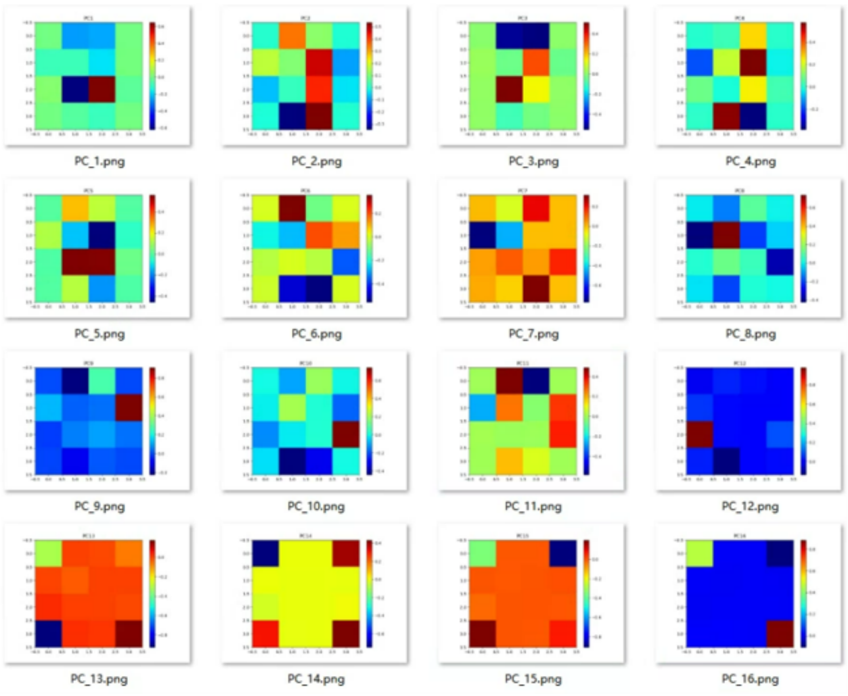

Figure 5.6.: 16 Basis of PCA

Then, we can put the same raw data set back into this PCA model to get its transformed version. The transformed data contains 16 values in an array. A transformed data may look like the following figure:

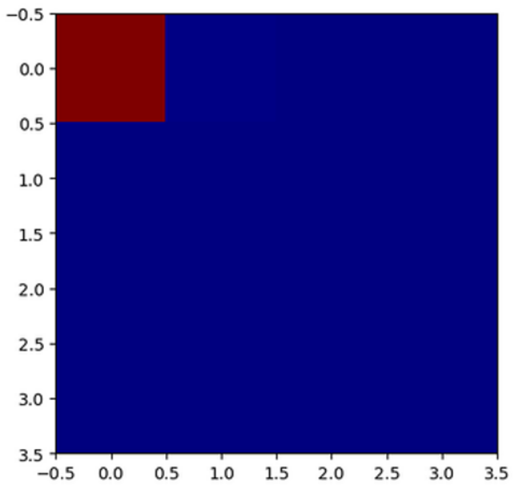

Figure 5.7.: Sample Transformed Data

The figure shows that this ample transformed data is consist of the first principal components of the 16 basis and no other basis. The 16 intensity values are fully transformed into the amplitude of 16 basis. We can repeat this process for the rest data sets to create their own PCA models.

After all of the raw data sets are transformed by their own PCA model, we need to reshape these data sets for the deep learning models. We have four types of models and they need different input and output forms.

On the input side, For an ANN model, it needs an n×m matrix as its input, where n is the samples number, and m is the number of eigenvalues. For a CNN model, it needs a 2-dimensional figure as input. Since we can simply reshape the matrices of ANN model into a figure when launching the CNN model, we do not need to prepare an extra input file for CNN models.

On the output side, for a classification model, we need to transfer the tilting angle into a bunch of labels with integer. We can use the label encoder method to create a simple map between tilting angles an the labels. For a regression model, we can directly leave the tilting angle as the output since a regression model is able to handle floating numbers.

Therefore, for each kind of data set we need to prepare three files for deep learning. The first file is an input file that contains 10000 samples with 16 amplitude values from PCA. The second file is a label file that contains the labels for 10000 samples. The third file is a tilt file that contains the tilting angles for 10000 samples. The labeled file can also be generated by a method in scikit-learn package:

```python
# Applying label encoder and save files
from sklearn import preprocessing

le = preprocessing.LabelEncoder()
le.fit(opt_le.reshape(-1))
opt_transformed = le.transform(opt_le.reshape(-1)).reshape(samples, 1)
opt_transformed = np.asarray(opt_transformed)
```

5.5 Deep Learning

From the previous section, we already knew that we have four deep learning models and these models can be categorized by the structure and the output of the model. To make it simple, we call ANN model as 1D model and CNN model as 2D model. In this section, we will introduce the detailed structure of our model.

5.5.1 1D-Classification

The 1-dimensional linear classification model is the first model of our study. The aim of the model is to return one of the ten labels after 16 eigenvalues are given. The structure of this model is as follows:

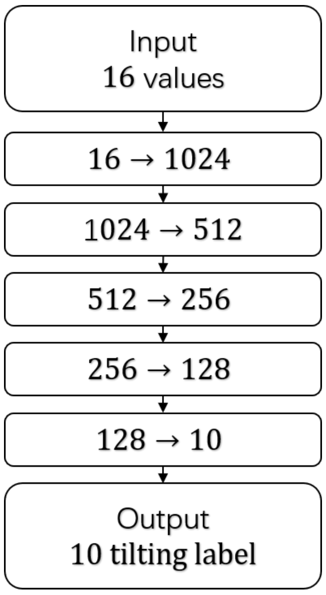

Figure 5.8.: Schematic of 1D-Classification

The 16 eigenvalues are first spread into 1024 nodes to increase the complexity of the model to a proper state. Then we let the number of nodes decreases layer by layer and come to 10 nodes after four layer. Those 10 nodes are correspond to the 10 labels which represent the tilting angles. Activation function SELU are applied to connect every two layer and we use CrossEntropyLoss for this model. The optimizer it uses is Adam and so do the other three models. We will use the accuracy of the labels to check if this classification model is learning and if it is valuable.

5.5.2 1D-Regression

The 1-dimensional linear regression model has almost the same structure as the previous classification model. The aim of this model is to return the tilting angle after 16 eigenvalues are given. The structure of this model is as follows:

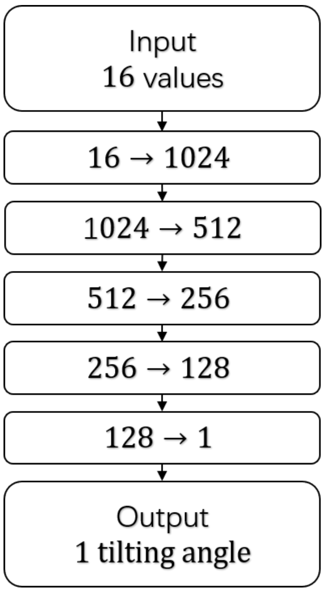

Figure 5.9.: Schematic of 1D-Regression

For this model, we use the same complexity with 1D-Classification model. The only difference between them is the output layer. Instead of dividing into 10 labels, the regression directly returns the tilting angle of the signal. Activation functions are still SELU function while the loss function is changed into MSELoss, which is a better choice for regression model. The output from this model will be a float number and it is impossible to return exactly the correct number with a deep learning model. Therefore, we do not use accuracy as our criteria to the model but use the results in the loss function. We check the order of magnitude for that result and calculate the difference range between the output and the true value. If the difference is equal or smaller than the precision of the tilting angle, we can say the model is valuable.

5.5.3 2D-Classification

The 2-dimensional classification model mixed by both convolutional layers and linear layers. Like the previous classification model, the aim of this model is to return one of the ten labels. However, the input layer takes images rather than matrix. We reshaped the image from 16 eigenvalues like the transformed data example in previous section. Its structure is as follows:

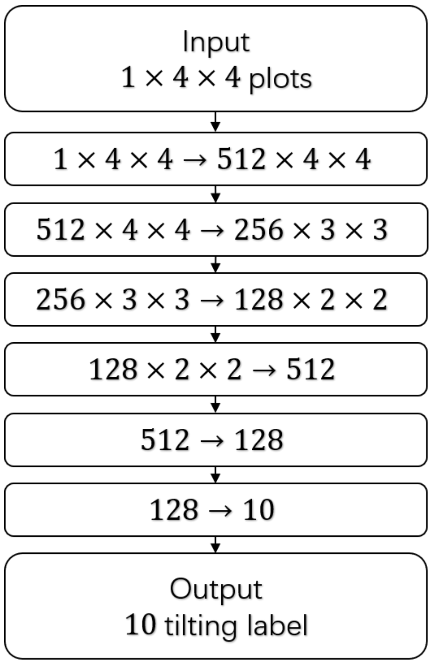

Figure 5.10.: Schematic of 2D-Classification

The reshaped input is in form of 4×4 image with one channel. Like what we did for 1-dimensional models, we increased the channel to 512, which allows the complexity of the model to raise to a proper level. Then, we gradually reduce the number of channels and use convolutional layer to converge the nodes at the same time. When the total values are reduced to 512, we flatten the 2-dimensional structure into 1-dimension. With a few linear layers, we reached 10 labels like 1D-classification model. Activation function and loss function of this model are the SELU function and CrossEntropyLoss respectively.

5.5.4 2D-Regression

The 2-dimensional regression model has almost the same structure as the 2D-classification model. The aim of this model is to return the tilting angle after an image is given. The structure of this model is as follows:

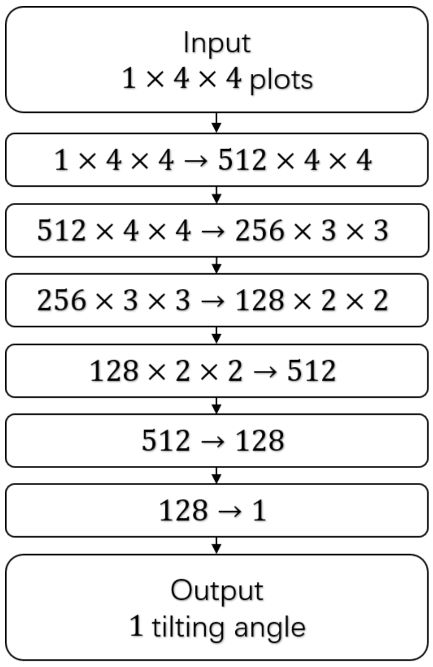

Figure 5.11.: Schematic of 2D-Regression

Our last model can be treated as it has the structure of the 2-dimensional classification model but with an output of 1-dimensional regression model. Its activation function are loss function are still SELU and MSELoss for the regression model. We use the difference from loss function to check if the model is learning and valuable.

5.6 Results

All of the data sets and deep learning models are now prepared. In this section, we will show the results for our study followed by the main steps. Note that all of the samples are generated by loops in Python, which means that these data are sorted by the tilting angle. We need to shuffle these files before passing them to the deep learning model. Our next step is to test the precision of our model. We use the simplest data set with no mirror map and test the 1D-classification model with various rotations. We find our model is workable for a bound range of 10^{-5} radian, with a precision of 10^{-6} radian. When we increase the precision to 10^{-7} radian, the model is no longer learning and return the result as follows:

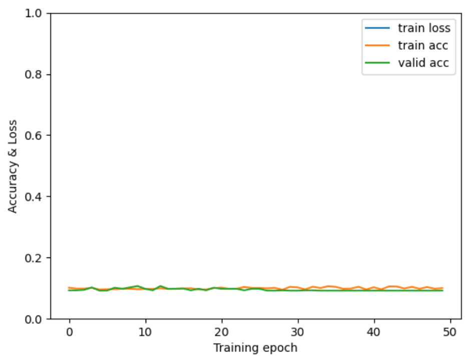

Figure 5.12.: Accuracy and loss on 10^{-7} radian

For the above figure we can see that the accuracy of both training and validation are about 0.1. It equals to the result that if we choose a label randomly from 10 choices. We found out for our current setting, the best precision is 1 micro radian. We generated all four kinds of data sets with a precision of 10^{-6} radian and put them into all of the models. We will talk about these results in the following paragraphs.

For the 1D-classification model, we trained the model with 50 epochs to achieve a stable outcome. The results are as follows:

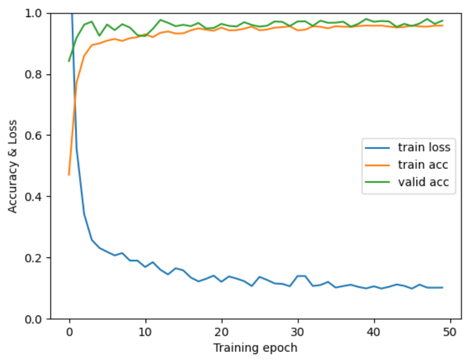

Figure 5.13.: Accuracy and loss for 1D-Classification nomap

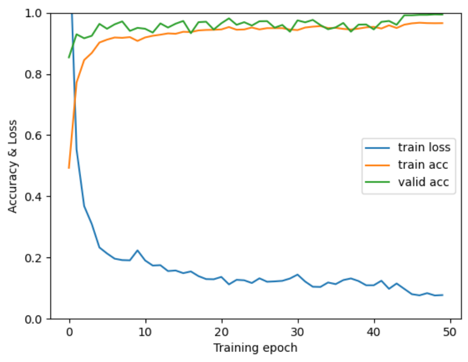

Figure 5.14.: Accuracy and loss for 1D-Classification map

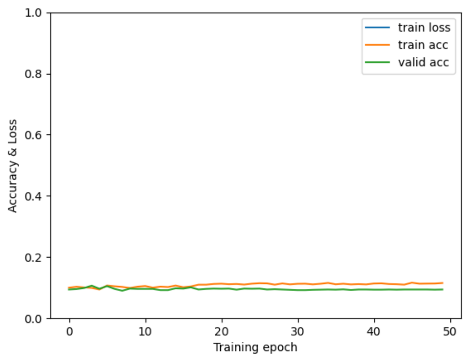

Figure 5.15.: Accuracy and loss for 1D-Classification random

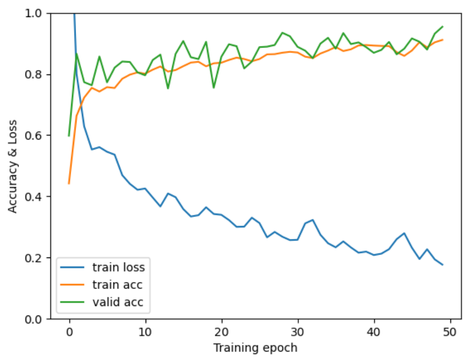

Figure 5.16.: Accuracy and loss for 1D-Classification miscenter

Except for the model trained by data with random mirror map, we have a quite successful training for the other three data sets. All three models could reach an accuracy over 90%. Especially for the mis-centered one, for this data set is the closest situation to gravitational wave detection in reality. After the taking the average accuracy over 5 training, the mis-centered model returns an accuracy of 91.24%. No over-fitting problem can be found from the line of validation accuracy. As for the model with random mirror map, there are several possibilities why the model failed to learn. We think the randomly generated maps have no connections to each other, so they might get messy when applying PCA method to these non-related data. We also consider that the model may not be able to find the center of the beam in a group of random maps, which made this data set become an impossible task.

For the 1D-regression model, we also trained the model with 50 epochs to make sure there is a stable outcome. Actually the model takes about 20 epoch and its result is as follows:

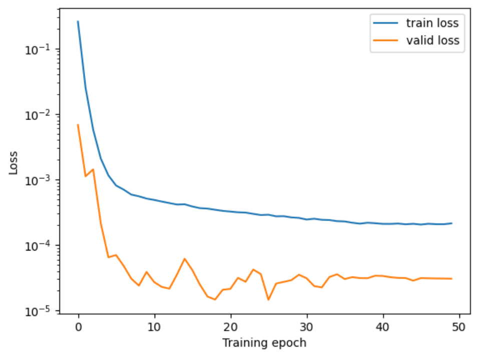

Figure 5.17.: Accuracy and loss for 1D-Regression nomap

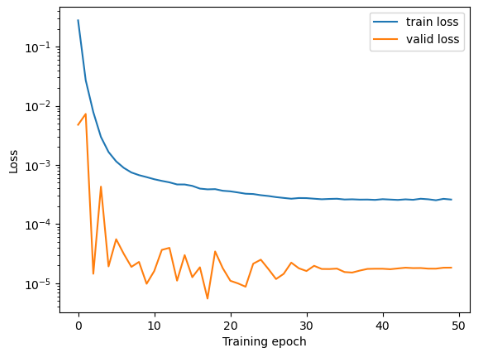

Figure 5.18.: Accuracy and loss for 1D-Regression map

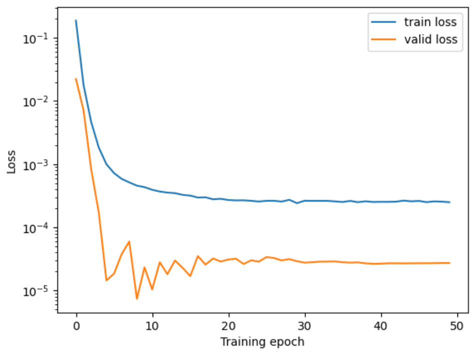

Figure 5.19.: Accuracy and loss for 1D-Regression random

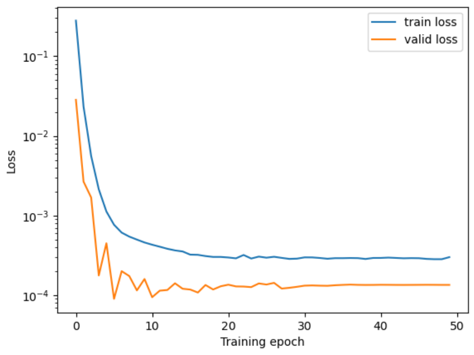

Figure 5.20.: Accuracy and loss for 1D-Regression miscenter

Since we use MSELoss for our regression models. MSELoss is calculated by square of the differences between the output and the true value. Therefore, if we take the square roots for the training and validation loss, it should be smaller than the chosen precision. From the figure we can see that there is an obvious decrease in the beginning, which means the regression model is learning. However, the loss varies from 10^{-5} to 10^{-3}. If take the square roots for the loss values, we will have a number around 10^{-2} radian, which is far bigger than our expected precision, which is 10^{-6} radian. We can make a simple conclusion based on the result. Based on the same complexity, the result of 1D-regression model does not match with 1D-classification model.

For the 2D-classification model, we also trained the model with 50 epochs to achieve a stable outcome. The results are as follows:

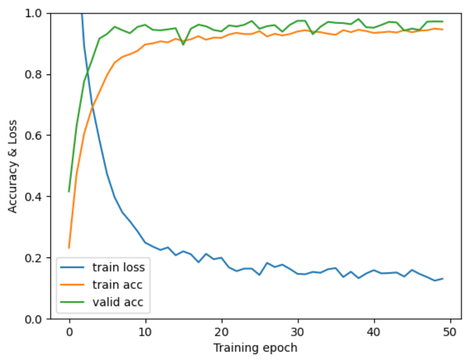

Figure 5.21.: Accuracy and loss for 2D-Classification nomap

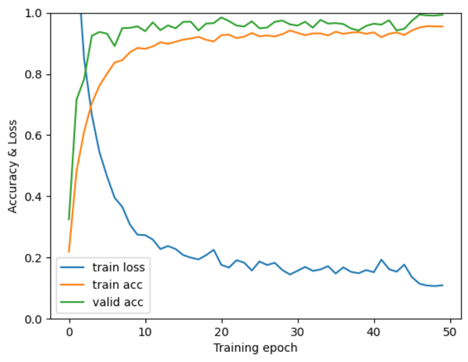

Figure 5.22.: Accuracy and loss for 2D-Classification map

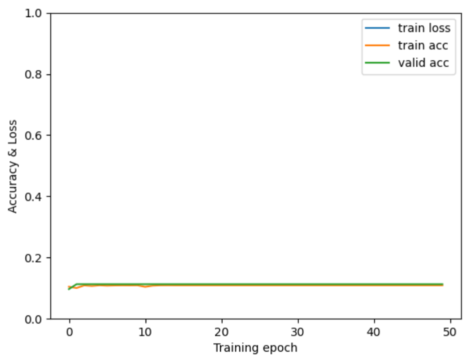

Figure 5.23.: Accuracy and loss for 2D-Classification random

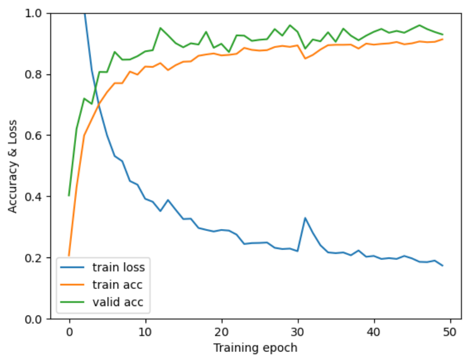

Figure 5.24.: Accuracy and loss for 2D-Classification miscenter

Most of the 2D-classification models are well trained after 50 epoch, except for the one with random mirror maps. We assume it is caused by the same reason as the 1D-classification model with the same input. The other results are also similar to what we have for 1D-classification model, the loss function is reduced to less than 0.1 and both training and validation results from the model gives us an accuracy over 90%. There is no obvious over-fitting problem from the validation accuracy. For the mis-centered model, we see a much stable line of training loss, which seems to be a good result. The average accuracy over 5 training for mis-centered model is 90.95%, which is close to 91.24% for linear models.

For our final model, the 2D-regression one, we also trained the model with 50 epochs to make a stable outcome. Except for the first training loss, the result seems to be the same as follows:

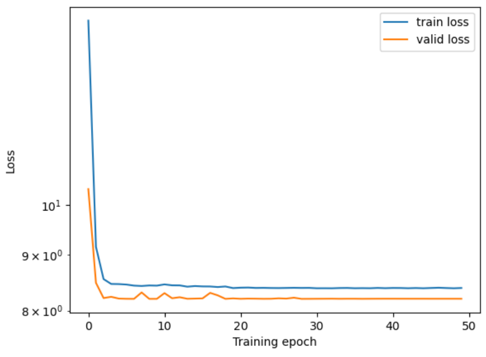

Figure 5.25.: Accuracy and loss for 2D-Regression nomap

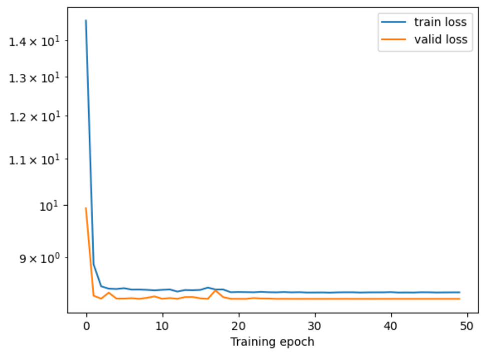

Figure 5.26.: Accuracy and loss for 2D-Regression map

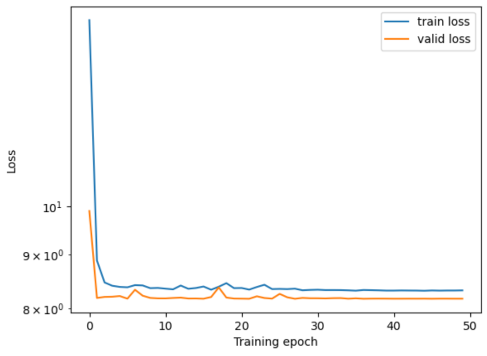

Figure 5.27.: Accuracy and loss for 2D-Regression random

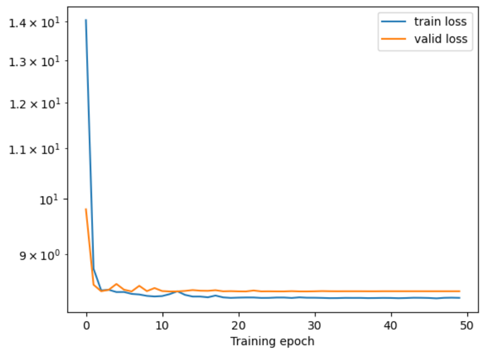

Figure 5.28.: Accuracy and loss for 2D-Regression miscenter

For our last model, we see that the training and validation loss are enormous compared to the 10^{-6} radian precision we have. What's worse is that except for the first training loss, the rest of the loss values look like a flat line. This feature means that the model barely learns nothing.

By far, we have 2 kinds of classification models that gave us quite successful results. Meanwhile 2 kinds of regression models does not work well with the same complexity. We need to step a little bit further to find out which of the classification model work better. A detailed training results for classification models can be found in the following tables:

Table 5.2.: Accuracy Table for Classification model

Mirror	ETM rotation							
Map-type	no-map		map		random		mis-center	
Model-type	1D-Cl	2D-Cl	1D-Cl	2D-Cl	1D-Cl	2D-Cl	1D-Cl	2D-Cl
Acc 1	0.962	0.970	0.962	0.967	0.110	0.111	0.919	0.909
Acc 2	0.959	0.970	0.959	0.971	0.110	0.111	0.906	0.914
Acc 3	0.961	0.971	0.963	0.970	0.106	0.109	0.902	0.915
Acc 4	0.957	0.970	0.961	0.966	0.108	0.118	0.926	0.902
Acc 5	0.964	0.967	0.960	0.971	0.124	0.110	0.909	0.908
Avg Acc	0.961	0.970	0.961	0.969	0.112	0.112	0.912	0.910

Both classification models have a stable result after several training. However, we still do not know which model is better only by the accuracy table. Here we introduced confusion matrices [35] to help us check the details within in these results. It is a well used table layout that allows visualization of the performance of a supervised model. It may also help us find out some hidden problems that can not be seen by accuracy. So we calculated the confusion matrices for these two models. The confusion matrices for the model without a mirror map are as follows:

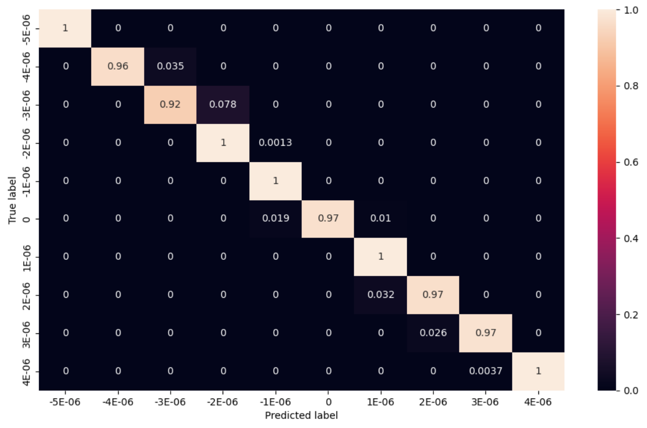

Figure 5.29.: Confusion Matrix for 1D-Classification nomap

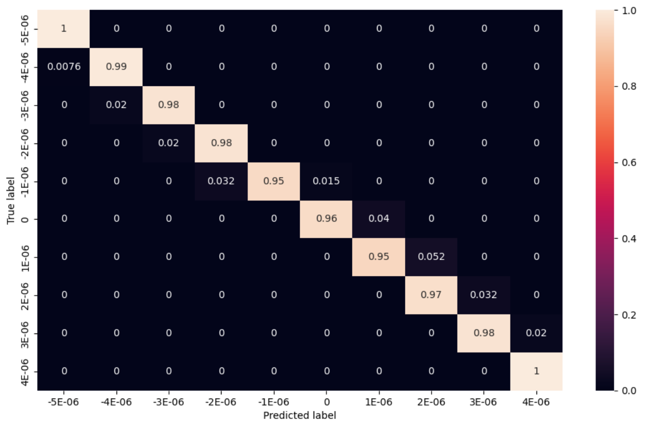

Figure 5.30.: Confusion Matrix for 2D-Classification nomap

For this group of data set, there is not much difference between two kinds of models. Those incorrect output only took a tiny part of the whole data sets and the loss are just some small mis-classification rather than outliers. In such case, following the output will not mis-leading the mirror too much. Since this is the simplest case of all four groups of training and can actually be solve by linear method in traditional way, we expect it to have the highest accuracy and stability. The result matched with our assumption.

Then, we keep moving on the next group of data sets, which is the group with a single mirror map. The results are as follows:

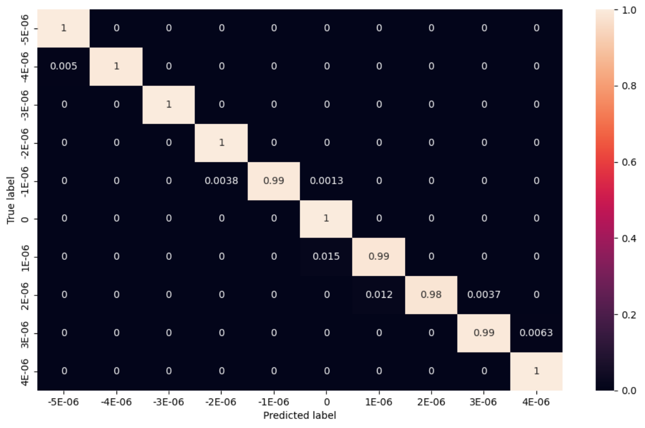

Figure 5.31.: Confusion Matrix for 1D-Classification map

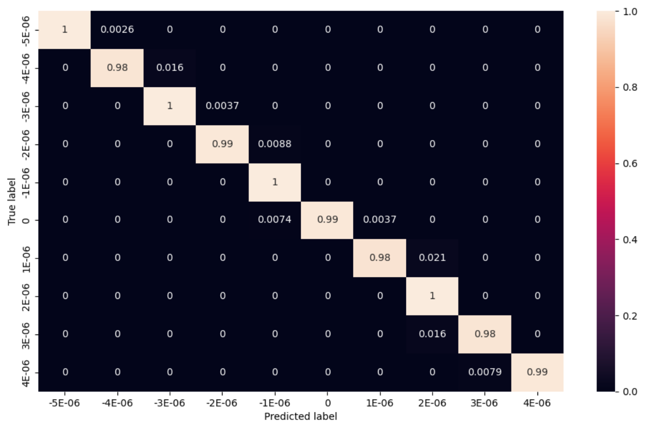

Figure 5.32.: Confusion Matrix for 2D-Classification map

In reality, adding a single map to the optical system does mess up the output. But as we expected, it does not affect the result from deep learning models. Because the roughness of the mirror is trained as a whole part, like adding a simple bias to the model without mirror map. The result also matched with our assumption.

We then skip the random group and jump into the mis-centered group because we already the random group of model fails to learn. The confusion matrices for the last group are as follows:

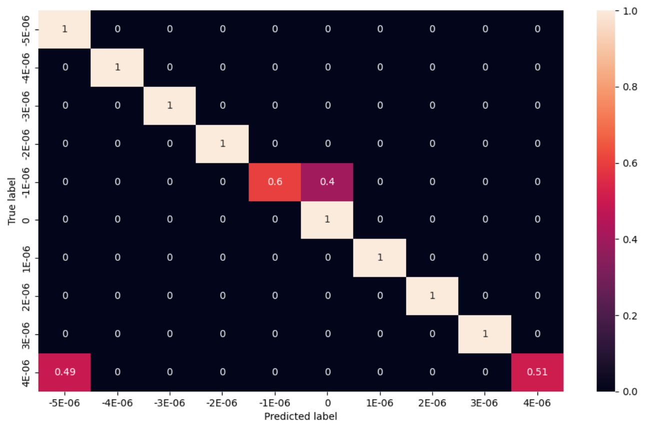

Figure 5.33.: Confusion Matrix for 1D-Classification miscenter

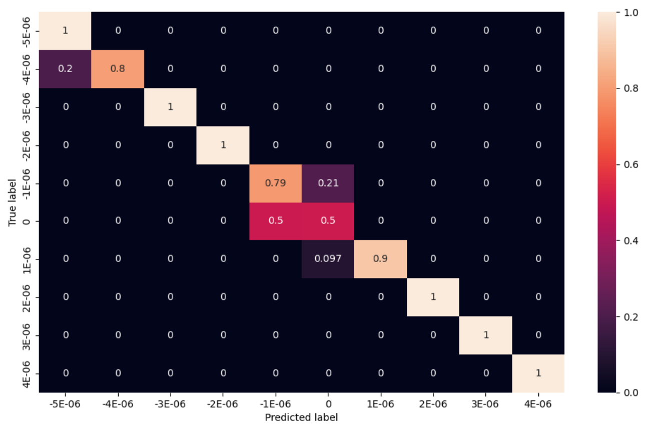

Figure 5.34.: Confusion Matrix for 2D-Classification miscenter

Here we see some interesting features. For the ANN model, we noticed that the model returns the correct label when there is a rotation applied to it. But the model has a poor accuracy on small rotations near the origin point. We also noticed that there is a high error rate at the upper boundary. This may happens because the lack of training data around that boundaries. However, this situation does not happens at the lower boundary. So we are not sure if it is this very reason that lead to this problem. For the CNN model, we see a slightly worse confusion matrix than the ANN model. The problem of poor accuracy around origin point still exist. While the boundary problem seems to affect on the lower bounds. To make sure if this is a normal problem, we did several extra training on both models. The result shows that sometimes this problem happens on upper or lower bound, and sometimes it happens on both sides. For a rare condition, the problem disappeared. The boundary problem randomly occurs shows a high opportunity that it is not the model that causes this problem, but the shuffled data sets. A solution would be extending the simulated data on the boundary. In addition, we check the average consumed by these two models:

Table 5.3.: Time Consumed for a Single Signal

	time (s)
ANN	1.7E-6
CNN	1.037E-4

The linear model is about a hundred times faster than using CNN models, which gives a faster speed of reaction when applied to data in real time. We can see both of the reaction speeds are in reasonable range.

We know the classification models work well on current data sets. Both ANN and CNN models are trained with a high accuracy at the precision of 10^{-6} radian. Though the accuracy around origin point is still a problem, we can say the model is quite stable at current state. We restricted the resolution of the signals to make sure it is possible to apply the model into real gravitational wave detection. But it is also possible that with a higher resolution, we will get a better precision for the model. Therefore, we remain the complexity of the model at the current state, changed the input resolution from 4×4 to 8×8, and increased the precision of the output. Our test begin with data sets that have a precision of 10^{-7} radian:

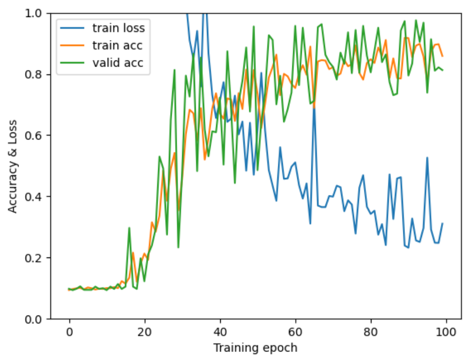

Figure 5.35.: Accuracy and loss for 1D-Classification on 10^{-7} radian

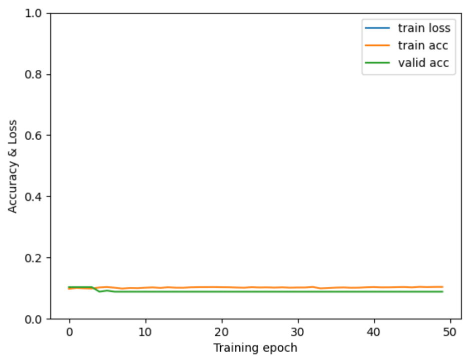

Figure 5.36.: Accuracy and loss for 2D-Classification on 10^{-7} radian

Remember at this precision, the 1D-classification model failed to train with an input of 4×4 signal. Now with a 8×8 signal, we see there is an obvious improvement in that model. Despite the 2D-classification model failed to train with an extra resolution, the 1D-classification model is actually learning something. Unfortunately, the model is unstable even when the epoch is increased to 100. We see the model reaches an accuracy around 85% to 90%, but with a strong fluctuation. During the current state, we cannot use such an unstable model.

We then tried training both of the classification models with a precision of $5 * 10^{-7}$ radian, which is a precision between this test and the previous data sets. We want to make sure if the previous 10^{-6} radian is already the best precision we can reach. This time, we only trained the model with 50 epochs to achieve a relatively stable outcome. The result is as follows:

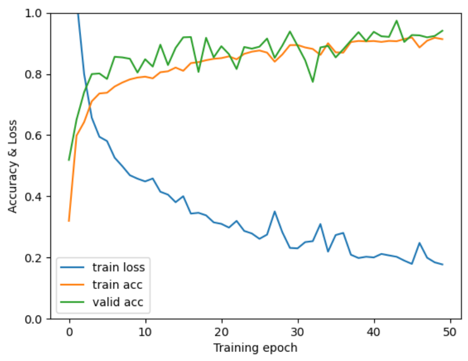

Figure 5.37.: Accuracy and loss for 1D-Classification on $5 * 10^{-7}$ radian nomap

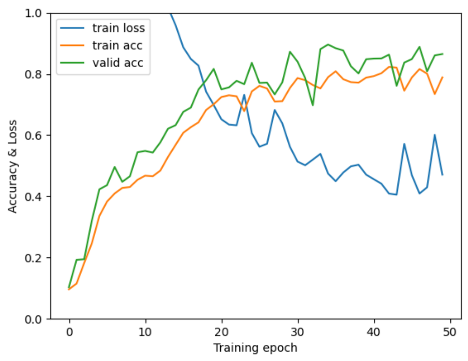

Figure 5.38.: Accuracy and loss for 2D-Classification on $5 * 10^{-7}$ radian nomap

This figure looks much better than the previous one. The accuracy of 1D-classification model is again stable around 90%. Its fluctuation is not that obvious like the result with finer precision. For 2D-classification model, we see the model started learning at this state. Although it only reaches an accuracy at about 85% and has some fluctuations, we will be glad to test on this model for a few more times.

So we generated the data set with single map and the data set with mis-centered map under the precision of $5 * 10^{-7}$ and started training with these new data sets. After training on both 1D-classification and 2D-classification models for five times, we took the average accuracy as what we did before. The following table shows the details of the results:

Table 5.4.: Accuracy Table for Classification model

Mirror	ETM rotation					
Map-type	no-map		map		mis-center	
Model-type	1D-Cl	2D-Cl	1D-Cl	2D-Cl	1D-Cl	2D-Cl
Acc 1	0.918	0.811	0.922	0.821	0.110	0.110
Acc 2	0.873	0.797	0.870	0.823	0.110	0.108
Acc 3	0.873	0.819	0.872	0.816	0.110	0.102
Acc 4	0.911	0.834	0.863	0.789	0.110	0.102
Acc 5	0.865	0.819	0.914	0.764	0.110	0.109
Avg Acc	0.881	0.816	0.888	0.803	0.110	0.106

From the table we can see that the ANN model works better than CNN model for the first group of models with no mirror map. The second group has nearly the same accuracy as the first group, as we expected. But both models failed with mis-centered data sets. Therefore, for the current model, the best precision is 10^{-6} radian.

Chapter 6

Summary

In this study, we tried to use deep learning technology to develop an automatic control system for mirror tilting in gravitational wave detection. All of our data and models are built on Python language, with the help of several related packages. We prepared dozens of data sets, all generated from a simulation program. Each simulation gave us the reflected signal from a Fabry-Perot cavity. And these data sets vary in the type of the mirror, precision of the tilting angle and the signal resolutions. Four deep learning models are built to compete with each other. To make sure the models to be stable and the results to be reproducible, we took average accuracy over five training for each model.

The best precision we can reach was 10^{-6} radian, with a 4×4 resolution. Among all of the four models, both classification models successfully obtained a high accuracy over 90% and was stable enough to be used. The 1D-regression model was learning, but it did not give us a valuable return. However, the 2D-regression model failed to learn anything and was the worst model among the four.

As for the mirror type of the data sets, both of the classification models have stable results under three groups of data sets. They both have high accuracy on the simplest data set with perfect flat mirror. They also kept that accuracy when we apply a roughness to that mirror as we expected. When we tried to generate random mirror maps and put them into one data set, the model fails to learn from the messy information. But the model works well when we added a mis-center factor to one rough mirror. Although we cannot apply our model to any random mirror, with the success of training on mis-centered data set, it is enough for us to apply the model to reality. Because in reality, the roughness of mirrors in a certain optical system usually does not change.

Furthermore, we tried some extra training rounds with a higher resolution of the signals. Although the resolution was restricted to 4×4 at the beginning, we still want to see if we could have a better model with more information. So, we choose to increase the resolution to 8×8. The information these data sets contains was four times larger than the previous ones. Unfortunately, with such resolution, the model still cannot achieve a precision of 10^{-7} radian. Even for a precision in the middle of two training round, $5 * 10^{-7}$ radian, the model fails at the mis-centered stage. Therefore, a precision of 10^{-6} radian was the best one we can have.

Comparing our model to the traditional control systems with linear calculation, the current precision is not enough. There are some future improvements that may help increasing the precision of our model. We can optimize the model by increasing the complexity of the model. The current complexity of our model is based on input resolution of 4×4. When we applied a much precise resolution, a much more complex model may do a better job. We can also optimize the model by increasing the labels of the models. It is hard for the model to jump from 10^{-6} to 10^{-7} radian, but it is possible to let the model achieve a valuable accuracy among 100 labels. Instead of changing the data sets, it is still possible to change the model itself to achieve the same goal. These are just two simple improvement that can be easily done under the current experiment. We think this is just a start of applying deep learning model in gravitational wave detector. For an optical system that is much more complex than a single Fabry-Perot cavity, deep learning technology may show its advantage on solving complex problems.

Appendix A

Structure of Finesse

```
base = f"""
    # ======== Constants ========================
    const fsb1 15M
    const mfsb1 -15M

    # ======== Input optics =====================
    l i1 1 0 n0
    s s_eo0 0 n0 n1
    mod eom1 $fsb1 0.15 2 pm n1 n2

    s s_eo1 0 n2 n3
    bs pickoff 0.0001 0.9999 0 45 n3 dump n4 n5
    s s_eo2 0 n4 n1ar

    m ARx 0 1 0 n1ar n1ar2
    s sAR 0 n1ar2 n6

    m ITM 0.996 4000e-6 0 n6 n7
    s cl 1 n7 n8
    m ETM 0.999945 5e-6 0 n8 n9

    s gouy_phase_tuner 0 n5 REFL
    attr ITM Rc 0
    attr ETM Rc 2

    cav cav1 ITM n7 ETM n8

    maxtem 5
    gauss* input i1 n0 0 2
    yaxis abs:deg
```

```
map ITM './mirror_map/mirror_map_{data_count}.txt'
knm ITM sub_map
conf ITM save_knm_binary 1
conf ITM interpolation_method 2
conf ITM integration_method 1

# ======== Output optics ===================
attr {variable_mirror} xbeta {tilt}
attr gouy_phase_tuner g 90 #optimized

beam ccd REFL
xaxis ccd x lin -2 2 {ccd_size-1}
x2axis ccd y lin -2 2 {ccd_size-1}
"""
```

Appendix B

Structure of Deep Learning

B.1 1D-Classification

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(ccd_size**2, 1024),
            nn.SELU(),
            nn.Dropout(p=0.1, inplace=False),
            nn.Linear(1024, 512),
            nn.SELU(),
            nn.Dropout(p=0.1, inplace=False),
            nn.Linear(512, 256),
            nn.SELU(),
            nn.Dropout(p=0.1, inplace=False),
            nn.Linear(256, 128),
            nn.SELU(),
            nn.Dropout(p=0.1, inplace=False),
            nn.Linear(128, 10),
        )

    def forward(self, x):
        logits = self.linear_relu_stack(x)
        return logits

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
```

B.2 1D-Regression

```
 class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(ccd_size**2, 1024),
            nn.SELU(),
            nn.Dropout(p=0.1, inplace=False),
            nn.Linear(1024, 512),
            nn.SELU(),
            nn.Dropout(p=0.1, inplace=False),
            nn.Linear(512, 256),
            nn.SELU(),
            nn.Dropout(p=0.1, inplace=False),
            nn.Linear(256, 128),
            nn.SELU(),
            nn.Dropout(p=0.1, inplace=False),
            nn.Linear(128, 1),
        )

    def forward(self, x):
        logits = self.linear_relu_stack(x)
        return logits

criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
```

B.3 2D-Classification

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.linear_conv_stack = nn.Sequential(
            nn.Conv2d(1,128,2),
            nn.SELU(),
            nn.Conv2d(128,256,2),
            nn.SELU(),
            nn.Dropout2d(p=0.3),
        )

        self.linear_selu_stack = nn.Sequential(
            nn.Linear(256*(ccd_size**2)*(ccd_size**2), 512),
            nn.SELU(),
            nn.Dropout(p=0.1, inplace=False),
            nn.Linear(512, 128),
            nn.SELU(),
            nn.Dropout(p=0.1, inplace=False),
            nn.Linear(128, 10),
        )

    def forward(self, x):
        x = self.linear_conv_stack(x)
        x = x.view(-1,256*(ccd_size**2)*(ccd_size**2))
        x = self.linear_selu_stack(x)
        return x

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
```

B.4 2D-Regression

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.linear_conv_stack = nn.Sequential(
            nn.Conv2d(1,128,2),
            nn.SELU(),
            nn.Conv2d(128,256,2),
            nn.SELU(),
            nn.Dropout2d(p=0.3),
        )

        self.linear_selu_stack = nn.Sequential(
            nn.Linear(256*(ccd_size**2)*(ccd_size**2), 512),
            nn.SELU(),
            nn.Dropout(p=0.1, inplace=False),
            nn.Linear(512, 128),
            nn.SELU(),
            nn.Dropout(p=0.1, inplace=False),
            nn.Linear(128, 1),
        )

    def forward(self, x):
        x = self.linear_conv_stack(x)
        x = x.view(-1,256*(ccd_size**2)*(ccd_size**2))
        x = self.linear_selu_stack(x)
        return x

criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
```

Acknowledgement

This study is written for my master degree and is based on my last year's study in Tokyo Institute of Technology. During these years, I received many helps and supports from my supervisor, my team mates of the deep learning study group and my colleagues in Somiya laboratory and my family.

First of all, I would like to express my deepest appreciation to my supervisor, Kentaro Somiya. He offered me the chance to join this outstanding laboratory and lead me through the gate of the fantastic astrophysics. He not only helped me built a steady foundation of basic theories in my area of study, but also invited me to join the deep learning study group so that I can get in touch with many other friends who shares the same interest. During my study, it is he who pointed out a clear path to me on how to choose a title, what need to be added into the experiment and where can I find related learning materials. Without his instruction, I could never undertaken this journey.

Many thanks to Hirotaka Takahashi from Tokyo City University. In our deep learning study group, he kindly shared many topics in the area of data analysis, machine learning and gravitational waves detection. We had several detailed discuss on how to build the models in deep learning. He explained the details of many data analyzing methods to me in my previous study, those knowledge helped me choose the method to use in this very study.

I have my special thanks to Kenichi Harada, the research lecturer of our laboratory. His lecture on development of laser detection gave me a better view to look through the history of laser physics. I also received many help from him in the laboratory during these years.

I would like to extend my sincere thanks to Seiya Sasaoka of our laboratory. We co-operated many times in many deep learning projects and other events. In this study, he helped me fixed many problems in those deep learning models and checked the typos and mistakes of the thesis for me. In the laboratory, his tutorial of python coding was really impressive and those skills I learned indeed helped me a lot in my study. In real life, he also supported me a lot when I first arrived Japan. His passion in deep learning and gravitational waves motivated me though out these years of study.

I am also thankful to Yuting Liu, who is a former student of our laboratory. From his study, I learned the basic ideas of deep learning and how deep learning technology can be apply in gravitational wave detection.

I'm also grateful to Koki Tachihara, who is also a former student of our laboratory. His previous study of using WFS signals to built a control systems for mirror tilting proved that the deep learning is workable in an optical system.

I send my appreciate to Naoki Koyama from the deep learning study group. His codes for generating mis-centered data sets was the simplest and fastest among several proposals. Without his help, I could not optimize my codes to the current stage.

Thanks should also go to Hayato Tanaka from our laboratory. Although we worked on different approaches of the new control system for mirror tilting, together we identify the same group of parameters in our data sets for comparison.

I'd like to acknowledge all other members of Somiya laboratory. I thank Sotatsu Otabe for announcing every seminar of the laboratory. I thank Homare Abe for sharing his view from the observatory. I thank Kaido Suzuki for his patients and guidance through the physics experiment. I thank Takanori Suzuki for his explanation on how to apply for various services of the university. I thank Wataru Usukura, Kotaro Takeguchi, Sakino Takeda and Daiki Haba for holding the weekly study event, we studied together and enjoyed ourselves during the event. I thank Ryo Iden and Kento Takeshita for giving their precious feedback on my deep learning presentation. I thank Diego Dominguez and Sapna Hassanaly, their joining to the laboratory promotes the exchange of different cultures, which is indeed lightening my university life.

Finally, I want to thank my parents. They offered me the financial supports I need through out these years. They also gave me many advises beyond university life. It does help me adapt to my life in Japan.

Once again I sincerely appreciate the above people and all other people that provided me their help. It is their kindly supports that helps me made it today.

Bibliography

[1] LIGO, https://www.ligo.caltech.edu/page/about, accessed: 2023-07-10.

[2] Virgo interferometer, http://public.virgo-gw.eu/virgo-in-a-nutshell/, accessed: 2023-07-10.

[3] KAGRA Observatory, https://gwcenter.icrr.u-tokyo.ac.jp/en/plan, accessed: 2023-07-10.

[4] L. Barsotti, M. Evans, and P. Fritschel, Alignment sensing and control in advanced LIGO, Classical and Quantum Gravity **27**, 084026 (2010).

[5] K. Tachihara, Posture control of mirrors using deep learning, Master's thesis, Tokyo Institute of Technology (2022).

[6] H. Tanaka, Development of mirror posture control method under the influence of non-uniform birefringence, Bachelor's thesis, Tokyo Institute of Technology (2023).

[7] D. D. Brown and A. Freise, Finesse, The software and source code is available at http://www.gwoptics.org/finesse.

[8] D. D. Brown, P. Jones, S. Rowlinson, S. Leavey, A. C. Green, D. Töyrä, and A. Freise, Pykat: Python package for modelling precision optical interferometers, SoftwareX **12**, 100613 (2020).

[9] A. Paszke *et al.*, PyTorch: An Imperative Style, High-Performance Deep Learning Library, in *Advances in Neural Information Processing Systems 32* (Curran Associates, Inc., 2019) pp. 8024–8035.

[10] W. G. Anderson and J. D. Creighton, *Gravitational-wave physics and astronomy: An introduction to theory, experiment and data analysis* (John Wiley & Sons, 2012).

[11] B. P. Abbott *et al.* (LIGO Scientific Collaboration and Virgo Collaboration), Observation of Gravitational Waves from a Binary Black Hole Merger, Phys. Rev. Lett. **116**, 061102 (2016).

[12] N. Andersson and K. Kokkotas, 8 Gravitational Wave Astronomy:The High Frequency Window (2005) pp. 255–276.

[13] O. D. Aguiar, Past, present and future of the Resonant-Mass gravitational wave detectors, Research in Astronomy and Astrophysics **11**, 1 (2011).

[14] K. Somiya, Detector configuration of KAGRA–the Japanese cryogenic gravitational-wave detector, Classical and Quantum Gravity **29**, 124007 (2012).

[15] Y. Zhao *et al.*, Frequency-Dependent Squeezed Vacuum Source for Broadband Quantum Noise Reduction in Advanced Gravitational-Wave Detectors, Phys. Rev. Lett. **124**, 171101 (2020).

[16] L. McCuller *et al.*, Frequency-Dependent Squeezing for Advanced LIGO, Phys. Rev. Lett. **124**, 171102 (2020).

[17] A. E. Siegman, *Lasers* (University science books, 1986).

[18] A. Perot and C. Fabry, On the application of interference phenomena to the solution of various problems of spectroscopy and metrology, Astrophysical Journal **9**, 87 (1899).

[19] R. Drever, J. Hall, F. Kowalski, J. Hough, G. Ford, A. Munley, and H. Ward, Laser Phase and Frequency Stabilization Using an Optical Resonator, Appl. Phys. B **31**, 97 (1983).

[20] E. D. Black, An introduction to Pound–Drever–Hall laser frequency stabilization, American Journal of Physics **69**, 79 (2001).

[21] H. Kogelnik and T. Li, Laser beams and resonators, Proceedings of the IEEE **54**, 1312 (1966).

[22] E. Morrison, B. J. Meers, D. I. Robertson, and H. Ward, Experimental demonstration of an automatic alignment system for optical interferometers, Appl. Opt. **33**, 5037 (1994).

[23] B. Swathi, ARTIFICIAL INTELLIGENCE : CHARACTERISTICS, SUBFIELDS, TECHNIQUESAND FUTURE PREDICTIONS, JOURNAL OF MECHANICS OF CONTINUA AND MATHEMATICAL SCIENCES **14** (2019).

[24] W. S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, The Bulletin of Mathematical Biophysics **5**, 115 (1943).

[25] D. George and E. A. Huerta, Deep neural networks to enable real-time multimessenger astrophysics, Phys. Rev. D **97**, 044039 (2018).

[26] E. Cengil, A. Çinar, and Z. Güler, A GPU-based convolutional neural network approach for image classification, in *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)* (2017) pp. 1–6.

[27] Convolutional calculation in deep learning, `https://axa.biopapyrus.jp/deep-learning/cnn/convolution.html`, accessed: 2023-06-20.

[28] Z. Zhang and M. R. Sabuncu, Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels, arXiv:1805.07836 [cs.LG] .

[29] S. Ruder, An overview of gradient descent optimization algorithms, arXiv:1609.04747 [cs.LG] .

[30] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, On the importance of initialization and momentum in deep learning, in *Proceedings of the 30th International Conference on Machine Learning*, Proceedings of Machine Learning Research, Vol. 28, edited by S. Dasgupta and D. McAllester (PMLR, Atlanta, Georgia, USA, 2013) pp. 1139–1147.

[31] J. Duchi, E. Hazan, and Y. Singer, Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, Journal of Machine Learning Research **12**, 2121 (2011).

[32] M. D. Zeiler, ADADELTA: An Adaptive Learning Rate Method, arXiv:1212.5701 [cs.LG] .

[33] D. P. Kingma and J. Ba, Adam: A Method for Stochastic Optimization, arXiv:1412.6980 [cs.LG] .

[34] F. Pedregosa *et al.*, Scikit-Learn: Machine Learning in Python, J. Mach. Learn. Res. **12**, 2825–2830 (2011).

[35] S. Stehman, Selecting and interpreting measures of thematic classification accuracy, Remote Sensing of Environment **62**, 77 (1997).