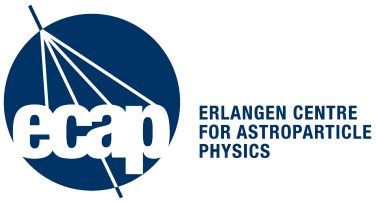
Photon flux calculation using Deep Learning

Master's Thesis in Physics

Presented by
Jigar K. Bhanderi
December 8, 2021

Erlangen Centre for Astroparticle Physics
Friedrich-Alexander-Universität Erlangen-Nürnberg

Supervisors:
Prof. Dr. Stefan Funk
Prof. Dr. Gisela Anton



Author:
Jigar Bhanderi

Supervisors:
Prof. Dr. Stefan Funk
Prof. Dr Gisela Anton

Institute:
Erlangen Center for Astroparticle physics
Department of Physics,
Friedrich-Alexander-Universitat Erlangen-Nurnberg

CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

ABSTRACT

Optical interferometry provides a sub-milliarcsecond resolution of astronomical objects. Intensity interferometry is a part of optical interferometry, where one correlates intensities of optical fluxes rather than amplitudes of the waves. In this case one records simultaneously the intensity of light from an object at different locations. For a successful measurement one needs a large collecting area for several telescopes separated by hundreds of meters and good time resolution of the intensity flux. Air Cherenkov telescopes, e.g., H.E.S.S. are a natural candidate for performing such a measurement. One of the important tasks is to determine the rate of photons hitting PMTs. For low rates, the individual pulses can be resolved and counted, but for high rates relevant for the IACTs the pulses from the photons overlap. We use several neural network algorithms (such as LSTM, Resnet, GRU) in order to determine the rate of photons hitting the PMT, including the high rates.

STATEMENT OF AUTHORSHIP

I, Jigar Bhanderi, born July 22, 1993 in Surat, declare that the work presented this thesis is entirely my own, and was done at Erlangen center for Astroparticle physics and as a part of Masters degree program at Friedrich-Alexander-Universität Erlangen-Nürnberg. This thesis includes no external published materials without assigning as reference. All the developments or ideas of other than myself are clearly marked.

Jigar Bhanderi Erlangen, December 8, 2021

ACKNOWLEDGMENTS

Firstly, I would like to express my sincere gratitude to Erlangen center for Astroparticle physics, where I got opportunity to work on this research project. I would like to thank all group members of Astro-Quantum optics who helped me during my work.

To begin with, my supervisor and co-supervisor, **Prof. Dr. Stefan Funk** and **Prof. Dr. Gisela Anton**, who provided me opportunity to do research on this particular topic, and provided me guidance for improving quality of results throughout this project.

Secondly, I would like to thank, **Dr. Dmitry Malyshev**, for helping me to resolve my queries and helped me to understand various concepts.

I am also thankful to **Andreas Zmija**, **Dr. Adrian Zink** and **Naomi Vogel**, for helping me in to understand data acquisition system and understanding of the instruments.

In addition to that, I would like to my family and friends, for providing me unconditional support during this academic program. A special thanks to Sharmila Rai, who always stood by my side and always encouraged me.

It suddenly struck me that that tiny pea, pretty and blue, was the Earth. I put up my thumb and shut one eye, and my thumb blotted out the planet Earth.

Neil Armstrong

1 INTENSITY INTERFEROMETRY

Our understanding of stars comes from high angular resolution results from star's light spectrum and variability, for which several attempts are done utilizing many advanced optical instruments including extremely large telescopes. The highest resolution so far is obtained by amplitude interferometers, with very long baseline, which are prefered to be operated in the near infrared since atmospheric turbulence have less effect on longer wavelengths. However, they are constrained by atmospheric turbulance. Intensity interferometry, an alternative approach to amplitude interferometry, is a method to achive higher angular resolution without results being affected by atmospheric turbulence. This chapter contatins different sections with theoretical formulation of intensity interferometry, first and second order correlation functions and signal to noise ratio calculation.

1.1 BRIEF HISTORY AND EARLY DEVELOPMENT

Measuring the angular diameter of is 400 years old problem. Galileo measured distance to the Vega and placed an upper limit on its angular size, by mentioning in his report the angular size of Vega as 5 arc-second with consideration of atmospheric scintillation. Newton also measured the angular size of stars and estimated angular size(upper bound) of Vega to be 2 miliarcseconds(mas), which is very close to actual angular size. By the late 1800 it was realized that the maximum angular resolution of around 1 miliarcsecond is determined by the atmospheric turbulence, instead of by the size of the aperture.

Revolution of interferometry began in 1868 when H. Fizeau proposed an interferometery technique, which is still the most powerful technique to measure angular diameter of very distant stars, with very high resolution. To understand Fizaeu's interferometer lets first understand Yong's double slit experiment. Consider a monochromatic light from distant a star falling on a screen with two pinholes P1 and P2, separated along the x axis. From Huygens-Fresnel principle, both pinholes will emit spherical waves which undergoes constructive and destructive interference with propagation lengths multiple of $(n + \frac{1}{2})\lambda$. The constructive interference is given by,

$$|P_1P| - |P_2P| = n\lambda \tag{1.1}$$

with n=0 $\pm$ 1, $\pm$ 2 etc. we find that,

$$|P_iP| = \sqrt{(x - x_i)^2 + (y - y_i)^2 + z^2} \tag{1.2}$$

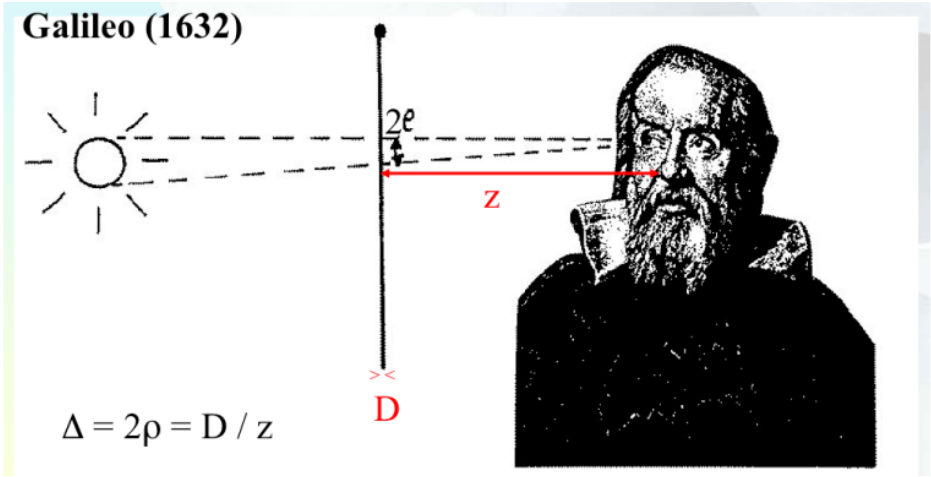


Figure 1.1: Galileo stood behind a rigid wire of a known thickness D suspended vertically. He determined the required distance z to completely obscure the star(Vega)by the wire. This method is also known as the stellar parallax method.

by rearranging,

$$|P_iP| = z(1 + \frac{(x - x_i)^2 + (y - y_i)^2}{2z^2}) \tag{1.3}$$

using equation 2.3 in equation 2.1 we obtain,

$$z(1 + (\frac{(x + B/2)^2 + y^2)}{2z^2}) - z(1 + (\frac{(x - B/2)^2 + y^2)}{2z^2}) = n\lambda \tag{1.4}$$

$$\frac{xB}{z} = n\lambda \tag{1.5}$$

or

$$\Phi = \frac{x}{z} = n\frac{\lambda}{B} \tag{1.6}$$

where Φ is the angular seperation between two successive maxima, which is independent of the coordinate y, and the reason of generation of bright and dark fringes. These fringes are oriented perpendicular to the line joining each other. Here, inter-fringe angular seperation can be given by, $\Phi = \frac{\lambda}{B}$. Now, for example, if distance between two holes is 1mm, and if $\lambda = 5500A$, then $\Phi = 113''$, which is just close to the human eye resolution limit. Fizeau realized this, and also determined that contrast of the interference fringes depends on the size and distance from the object, and accodring to that it widens.

To get more understanding, of how fringes depend on distance, lets consider whole picture geometrically . Consider two incoherent point like composite source instead of one coherent source in Young's double slit experiment. Both are separated by an angle Δ. Thus, there will be superposition in the plane of the observer between fringes, which are separated by an angle Δ. If $\Delta = \Phi/2$, then both of the fringes would exactly overlap each other and cancel each other by vanishing resulting contrast. The quantity ν in figure 2.2 is known as 'visibility' of star. If the star is resolve, i.e, $I_{min} = 0$, then visibility is 1, or in other case visibility is 0 since $I_{max} = I_{min}$. Fizeau applied this idea and used masking of

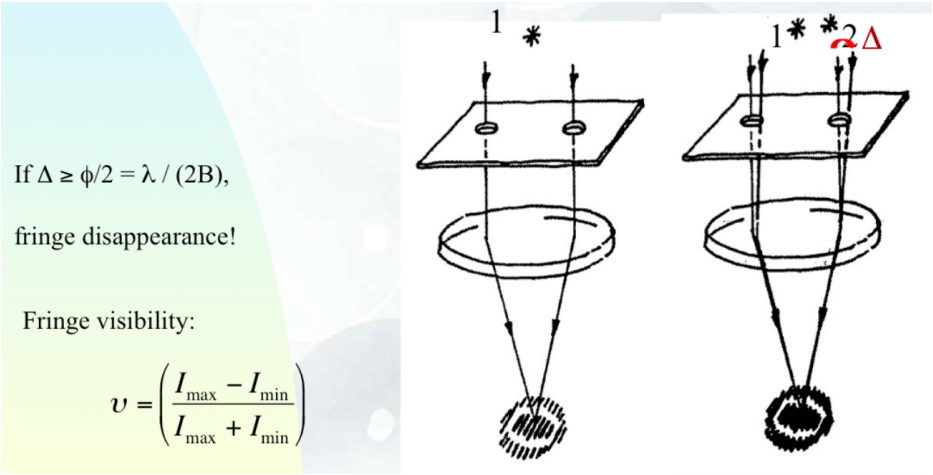


Figure 1.2: Consideration of two composite incoherent source fringes in young's double slit experiment with two slits with distance Δ, cancels resolutions created by individual fringes

elongated aperture at the front of the telescope in order to adjust the distance between two apertures until visibility of fringes vanishes. Around 1874, M. Stephan implemented this experiment with 80 cm telescope, and saw generated fringes in the stellar observation, and provided angular resolution upper limit by an order of 0.16 arc-seconds for stars. Later on, Michelson and Pease, in 1920, at Mt. Wilson had performed first successful measurement of angular diameter of star, which is considered as birth of high resolution astronomy[1]. With 36" telescope, Michelson measured the angular diameter of Jupiter's Galilean moons. In 1920, they also measured angular diameter of the Berelgeuse(47 mas) with 6m of interferometer baseline[1]. In the preliminary experiments of Michelson, he noticed the drift in fringes, which occurred due to atmospheric turbulence effects, inducing path difference between two beams. Michelson measured this drift time scale as of order of milliseconds. In 1930, Pease developed 15m version of their previous imterferometer, but unfortunately the experiment failed to provide convincing results and indicated for the need of very high precision optics, and then was abandoned. Later on, it took around 30 years for next breakthrough in optical interferometry.

First attempt to investigate coherence theory, was by Verdet in 1865, who measured coherence(transverse) length of sun as 0.1mm by measuring angular diameter of sun[2]. The crucial time for the development of coherence theory was in between 1920 and 1960. In this duration several notorious scientists, such as, Hopkins, Wolf, Weiner, Van Cittert, Zernike, have investigated and outcome of their efforts was quantification of correlation between space-time fluctuations and degree of coherence, which is obtained by Fourier transform of radiant distribution of star, which is known as Cittert-Zernicke theorem, and was first applied in radio astronomy by M.Ryle in 1952[3]. By the time radio interferometry had implemented modern techniques and improved understanding of coherence theory, which gave boost for development of intensity interferometry, and provided most highest angular resolution images till today.

From experiments, R. Hanbury Brown realised that patterns recorded at two points are correlated .i.e, low frequency intensity fluctuations were correlated, which gives advantage

over necessity of synchronised oscillators at two distant receivers, and no longer required with advantage of longer baselines. Along with R. Twiss, foundation of theory of intensity interferometry was developed by R. Brown, and along with that they built the first radio intensity interferometer, through which they succeeded to measure the expected correlation and angular size of stars (Cygnus A and Cassiopeia A)[4].They noticed the biggest advantage of intensity interferometry, in which correlation is not affected even though the source is highly fluctuating due to atmospheric turbulence. Later on, they both preformed series of laboratory experiments for correlation analysis, in which they used two PMTs placed few meter away from pinhole through which narrow bandwidth thermal light pass from source, and they measured the correlation as a function of detector separation, and then attempted to measure intensity fluctuations while measuring diameter of Sirius A [5]. The result from this experiment was surrounded with many problems, since there were many experiments which followed this idea but were performed with different bandwidth of light(specially broadband light source), which corresponds to low spectral density, which requires extremely large integration time to detect significant correlations(Brown and Twiss, 1974)[6]. This was confirmed several times(eg, Morgan and Mandel, 1966[7]), and quantum understanding of Hanbury Brown's and Twiss's work developed quantum optics. For the constriction of intensity interferometer they used two large optical telescopes, and since measurements were insensitive to the atmospheric fluctuations, they didn't necessarily required high precision optics but simple detectors. The entire setup was established in Narrabri(Australia), with circular tracks on which there are two movable reflectors placed, and the control center was located in the center where signals were correlated. The intensity interferometry experiment measured the angular diameters of 32 stars in between 1965 and 1972. Narrabri interferometer measured angular diameter of the southern hemisphere stars(specially brighter stars), till the sensitivity limit of the instrument had been reached. Therefore, projects with larger interferometer size were abandoned, since with the use of amplitude interferometry one could achieve similar sensitivity in the 1/40th time of intensity interferometry.

1.2 THEORY OF COHERENT LIGHT

1.2.1 *classical interpretation*

Interference can be interpreted in terms of correlation. Measurements of deviations from distant point like source, requires consideration of originated waves as planar waves, which are result of superposition of multiple monochromatic plane waves, with certain phase and has same frequency over time. Thus, the electric field amplitude of these plane waves can be expressed as,

$$E(t) = \int E(\nu) e^{i\nu t} d\nu \tag{1.7}$$

Two sources are coherent in nature, if waves originated from them have identical frequency and waveform, and such special case is known as coherence. With the time one also require to know about how well the phase of waveform is, which can be determined

by spatial coherence. Spatial coherence is correlation calculation of wave's phase at different points along the perpendicular to the wave propagation direction. But, calculation of phase at different points along the propagation direction, helps to determine how well monochromatic waves are, which is known as Temporal coherence (**coherent time** τ_c, and temporal **coherence length** l_c)[2].

$$\tau_c = \frac{1}{\Delta\nu} \tag{1.8}$$

where, $\Delta\nu$ is bandwidth of light, with $c = \lambda\nu$ coherence time can also be determined in terms of $\Delta\lambda$(wavelength bandwidth). For very small bandwidth,

$$\tau_c = \frac{\lambda_0^2}{c\Delta\lambda} \tag{1.9}$$

Similarly, at certain time from different angular positions θ, electric field of emitted superimposed plane waves can be given by,

$$E(x) = \int E(\theta) e^{ikx\theta} d\theta \tag{1.10}$$

There is also variation in measurement of x, Δx, depending on electric field variation over variations over angles $\Delta\theta$, and it occurs with relation,

$$\Delta\theta \sim \frac{1}{k\Delta x} \tag{1.11}$$

Another interesting quantity is coherence volume, which can be obtained by coherence cross-section area (Δx^2)[1] and coherence depth(length), within which classical EM-field is well-defined, and can be given by,

$$\Delta V \sim \Delta x^2 \Delta l \sim \frac{\lambda^4}{\Delta\theta^2 \Delta\lambda} \tag{1.12}$$

1.2.2 *Quantum description*

Connection of coherence with quantum mechanics can be understood by the use of uncertainity principle for phase-space calculation, in another words coherence volume

$$\Delta V \sim \frac{h^3}{\Delta p_x \Delta p_y \Delta p_z} \tag{1.13}$$

Since photons within this region have same states, they are indefinite(Mandel, Wolf, 1995). For the distant source, with small angular size $\Delta\theta$,

$$\Delta p_x = \Delta p_y \sim p\theta \sim \frac{h}{\lambda}\theta \tag{1.14}$$

1 Δx is known as transverse coherence length within which the wave is approximately planar and Δx^2 is crossectional area for coherence which linearly depends on deviations in θ

and,

$$\Delta p_z \sim \frac{h}{\lambda^2}\Delta\lambda \tag{1.15}$$

the phase space volume can be determined as,

$$\Delta V \sim \frac{\lambda^4}{\Delta\theta^2\Delta\lambda} \tag{1.16}$$

In the quantum description, photons within this region are correlated, even though they are indistinguishable, because they are in same state, which was realized by Hanbury Brown and Twiss (1957).

1.2.3 *Connection with correlation*

Correlation helps to understand interference phenomena more precisely. If the time delay between two two beams in space and time is smaller than coherence time, then they are highly correlated. If the time delay of the order of coherence time is introduced, then they no longer are correlated by having different frequency and phase (Mandel, Wolf, 1995; Labeyrie et al, 2006). Thus, correlation is also the tool to describe the coherence effect. There are several correlation functions, but specially two are very interesting, which are, first order correlation function g^1[8] and second order correlation function g^2.

$$g^{(1)}(r_1, r_2, \tau) = \frac{\langle E^*(r_1, t) \cdot E(r_2, t_1 + \tau)\rangle}{\langle E^*(r_1, t) \cdot E(r_1, t)\rangle} \tag{1.17}$$

,where r_1, r_2 are two positions in space and τ is time difference. The electric field averaged over time(relatively large compare to τ) for particular position can be given by,

$$\langle E(t)\rangle = \frac{1}{T}\int_0^T E(t)dt \tag{1.18}$$

Thus, time averaging makes $g^{(1)}$ function only dependent on τ, $g^{(1)}$ function, compares the electromagnetic field amplitude at given two different point with given time difference τ. Denominator in $g^{(1)}$ function, is for normalization, which is independent of position preference.

On the other hand, only $g^{(2)}$ function is of our interest because intensity interferometers measures intensities instead of amplitudes [8] , $g^{(2)}$ function can be measured by photo-detectors at different positions, and can be given by,

$$g^{(1)}(r_1, r_2, \tau) = \frac{\langle E^*(r_1, t)E^*(r_2, t_1 + \tau) \cdot E(r_1, t)E(r_2, t_1 + \tau)\rangle}{\langle E^*(r_1, t)E(r_1, t)\rangle \cdot \langle E^*(r_2, t)E(r_2, t)\rangle} \tag{1.19}$$

from $I = E^*E$

$$g^{(1)}(r_1, r_2, \tau) = \frac{\langle I(r_1, t) \cdot I(r_2, t + \tau)\rangle}{\langle I(r_1, t)\rangle \cdot \langle I(r_2, t + \tau)\rangle} \tag{1.20}$$

, where I is intensity at different positions, and thus $g^{(2)}$ can be measured by probing intensities. And by considering intensity fluctuation around average intensity $I(t) = \langle I(t) \rangle + \Delta I(t)$, $g^{(2)}$ function can be given by,

$$g^{(2)} = 1 + \frac{\langle \Delta I_1 \cdot \Delta I_2 \rangle}{\langle I_1 \rangle \cdot \langle I_2 \rangle} \tag{1.21}$$

Thus, for different time intensity(intensity fluctuations) products at two positions, the $g^{(2)}$ function also would be different,

- $\langle \Delta I_1 \Delta I_2 \rangle = 0$: random intensity fluctuations, fluctuations get average out, and $g^{(2)} = 1$, coherent light[8]
- $\langle \Delta I_1 \Delta I_2 \rangle > 0$: $g^{(2)} > 1$ no average out of intensity fluctuations, which makes light chaotic, which can be similarly detected at both detectors mostly. At smaller time difference photons are bunched, which if central part of investigation in research in Intensity Interferometry[9]
- $\langle \Delta I_1 \Delta I_2 \rangle < 0$: $g^{(2)} < 1$ Anti-bunching of photon occurs, $g^{(2)} < 1$, both detector detects photon irregularly due to not averaging out of intensity fluctuations, in other words very often one detector detects photon while other not[9]

For the case of thermal light source with atoms emitting electromagnetic waves towards the direction of observer and undergo to interference in that particular direction. The relation between both these correlation function is known as **Siegert relation**[8].

$$g^{(2)}(r_1, r_2, \tau) = 1 + |g^{(1)}(r_1, r_2, \tau)|^2 \tag{1.22}$$

From the shape of the source the $g^{(1)}$ function is easily determined, $g^{(2)}$ function can be determined by the Siegert relation.

1.3 RELATION BETWEEN SOURCE STRUCTURE AND LIGHT COHERENCE

1.3.1 *Van-Cittert-Zernike Theorm*

For the distant source, emitting planar wave and extended incoherent light, source intensity distribution can be understood by $g^{(1)}$ function, and the theorem representing this relation is known as **Van Cittert-Zernike Theorm**[2]:

$$g^{(1)}(r_1, r_2) = e^{ik(r_2 - r1)} \frac{\int_\sigma I(r) e^{-ik(s_2 - s_1)r} d^2r}{\int_\sigma I(r) d^2r} \tag{1.23}$$

As mentioned in Fig. 1.3[2], incoherent light source is denoted as σ, with origin O, with intensity $I(r')$ at distance r. Two detectors, $P_1(r_1)$ and $P_2(r_2)$, are located at distance $r_1 = r_1 s_1$ and $r_2 = r_2 s_2$. Using Eqn. 1.23, which determines that the g^1 function is proportional to the two-dimensional Fourier transformation of light intensity distribution of source, and allows us to probe geometry of the source(used in amplitude interferometry). As

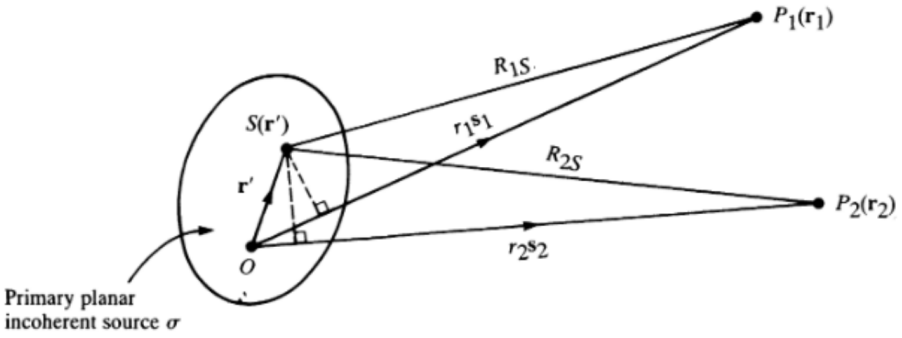


Figure 1.3: Graphical interpretation of the Cittert-Zernike theorm.

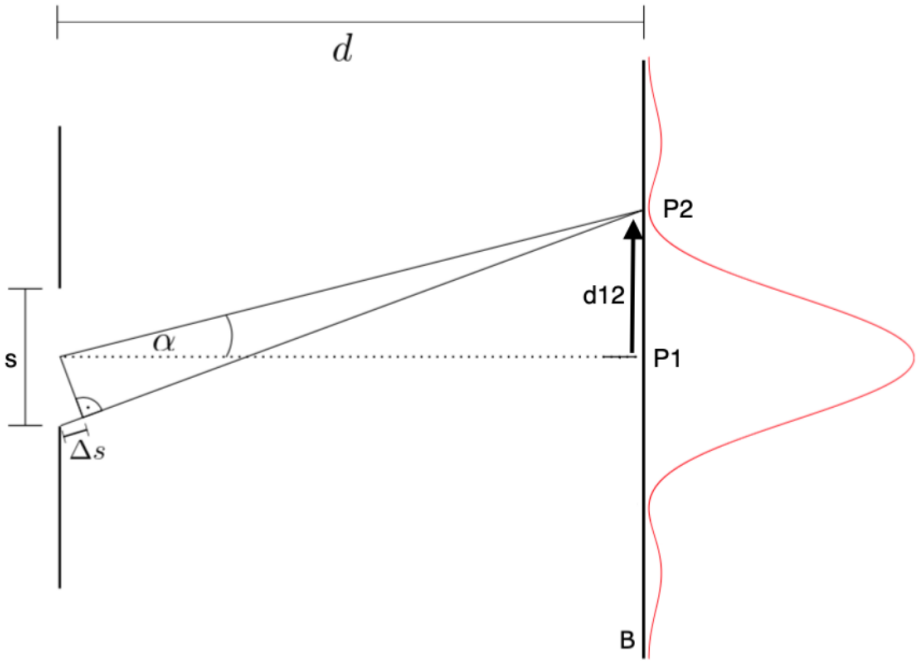


Figure 1.4: Interference pattern is detected at the observer plane using which slit geometry can be determined.

mentioned above, single slit experiment with slit width of s creates interference. These interference patterns can be determined at the observer plane with two detectors(P_1 and P_2) separated with distance d_{12}, as shown in Fig. 1.4, and are located in such a way that detector P_1 is at first maxima and detector P_2 is at first minima of interference pattern. Distance between slit and observer plane is d and angle between center of slit and source P_2 is α. The distance between two detectors can be determined as,

$$tan(\alpha) = \frac{d_{12}}{d} \Rightarrow d_{12} = tan(\alpha) \cdot d = \alpha \cdot d \Rightarrow d_{12} = \frac{\lambda}{s} \cdot d \tag{1.24}$$

Therefore, from Eqn. 1.24 it is clear that the for large wavelength and large distance between detector and source, it is more like to have coherent light, but with also decrease

in slit size coherent nature of the light decrease. Now, with this consideration but with circular geometry of slit, one can rewrite $g^1(d_{12})$ as [8],

$$g^1(d_{12}) = \frac{2J_1(\pi s \frac{d_{12}}{d\lambda})}{\pi s \frac{d_{12}}{d\lambda}} = \frac{2J_1(X)}{X} \tag{1.25}$$

where, J_1 is Bessel function.

1.3.2 *Wiener-Khinchin Theorm*

Another important relation between both first order and second order correlation function is **Wiener-Khinchin Theorem**, which can be obtained by process mentioned below. Consider normalized spectral intensity distribution $F(\omega)$, is given by Fourier transform of first order correlation function $g^2(\tau)$,

$$F(\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty} g^1(\tau)e^{i\omega\tau}d\tau \tag{1.26}$$

likewise with inverse Fourier transform leads us to determination of first order correlation function $g^1(\tau)$:

$$g^1(\tau) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)e^{i\omega\tau}d\tau \tag{1.27}$$

1.4 SIGNAL TO NOISE RATIO CALCULATION

The main motivation behind this project is to improve signal-to-noise ratio[10].For the calculation of SNR(signal to noise ratio) in intensity interferometry, observable, $g^2(r_1, r_2, \Delta t)$(equation 1.20) is the observable and can be further factorized into it's spatial and temporal parts,

$$g^2(r_1, r_2, \Delta t) = 1 + g_r^2(r_2 - r_1)g_t^2(\Delta t) \tag{1.28}$$

Spatial correlation part is related to source geometry. Square of the Fourier transform of the source angular size is proportional to the spatial correlation[2, 11], and with the help of telescope array's very sensitive to temporal correlation, it can be measured accurately. Thus, for the improvement of temporal correlation calculation, we select $r_1 = r_2$ and the Eqn. 1.28 solely depends on temporal correlation part, which is,

$$g^2(r_1, r_2, \Delta t) = 1 + g_t^2(\Delta t) \tag{1.29}$$

In this case, as mentioned in § 1.2.1, there will be strong correlation for small optical bandwidth of Δ, and the $g_t^2(\Delta t)$ is the Fourier transform of optical spectra S[11]. The time integrated signal in terms of temporal correlation can be given as,

$$S = \int_{\frac{-t_0}{2}}^{\frac{t_0}{2}} g^2(\Delta t) d\Delta t \tag{1.30}$$

but for second order correlation function for optical bandwidth $\Delta\nu$ can also be written as[9, 11],

$$g^2(\tau) = 1 + e^{-2\frac{|\tau|}{\tau_c}} \tag{1.31}$$

, where coherence time $\tau_c = \frac{1}{\Delta\nu}$. For the astronomical source correlation time is of the order of picoseconds. Thus, there is smearing of correlation signal. This smearing is determined by electronic time resolution τ_e. From this the smeared signal amplitude can be calculated with the help of τ_e and g^2 function:

$$A = \frac{1}{\tau_e} \int_{\frac{-\tau_e}{2}}^{\frac{\tau_e}{2}} g^2(\Delta t) d\Delta t = \frac{\tau_c}{\tau_e} \tag{1.32}$$

In addition to that, for observation time T and detection efficiency of the system ϵ, statistical fluctuations in intensity correlation can be determined as,

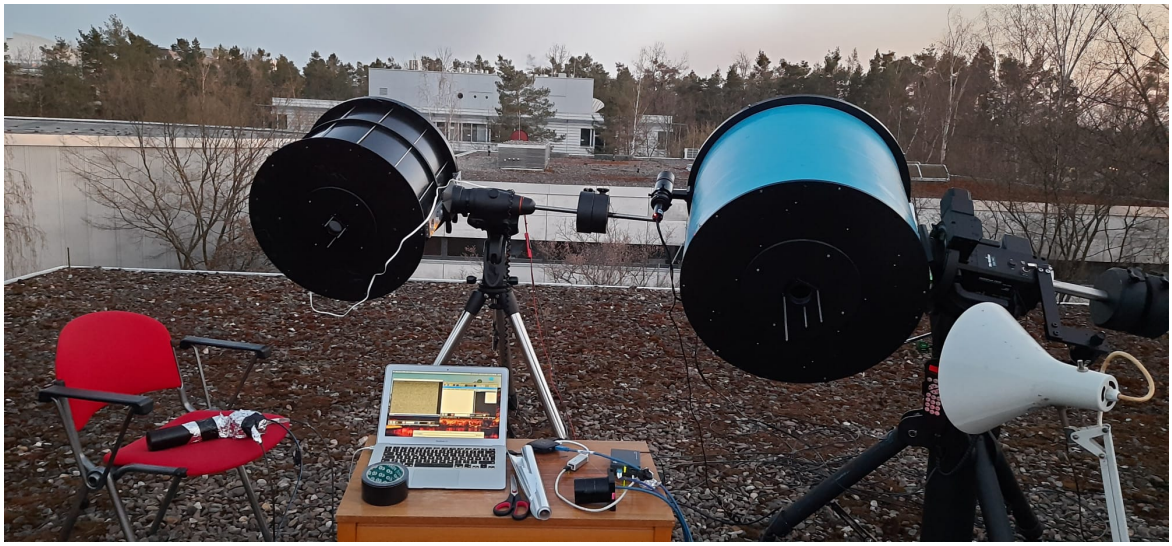
$$N = \frac{1}{R}\sqrt{\frac{\tau_e}{T}} = \frac{1}{C\epsilon n(\nu)\Delta\nu}\sqrt{\frac{\tau_e}{T}} \tag{1.33}$$

where, $n(\nu)$ is number of photons per optical bandwidth, C is cross-sectional area of telescope for photon collection, Δnu is optical bandwidth, R is photon rate($R = \sqrt{R_1 R_2}$). For the improvement of SNR, the accurate photon bunching and photon rate determination are important. Therefore in this thesis for the rate calculation, first the deep learning method(which is explained in detail in next chapter) is used, first for laboratory experiment, and then used for rate measurement of Sirius using IceAct telescope.

H.E.S.S(High Energy Stereoscopic System) is an IACT(Imaging Atmospheric Cherenkov Telescope), which is mainly used to study cosmic rays of the energy range GeV to TeV. They are located in Namibia, near the Gamsberg mountain. As shown in Fig. 1.5 there are four telescopes of Phase I established in 2004 by making square like geometrical arrangement, having diameter of 12m, and in order to make it more capable at lower energy analysis, Phase II telescope(28m diameter), has been added, which is bigger than Phase I telescopes. H.E.S.S experiment is international collaboration and includes 40 institutes from 13 different countries[3]. For Phase I telescopes total mirror area per telescope is 108 m^2, while for fifth telescope it is $614m^2$. Phase I adjacent telescopes are located at 120m from each other, and each Phase I telescope have around 382 facet, aligned to detect object at distance 10km apart, which is in general height of an air shower. Primary goal of the experiment is study of non-thermal universe.

3 https://www.mpi-hd.mpg.de/hfm/HESS/pages/about/

(a) H.E.S.S. telescope



(b) IceAct telescope arrangement at rooftop of our institute, used for correlation analysis and rate measurement from Sirius

Figure 1.5: H.E.S.S. telescopes with four phase-II telescopes and one phase-I telescope located in the middle[2], and IceAct telescopes

While using H.E.S.S like bigger telescopes, due to their large mirror area, upto 1GHz or high photon rate is expected to be detected. This kind of measurements comes with overlapping of photons. To determine each photon, one can use peak find method, but it has limitation that it can only extract physically appeared peaks, which makes it working well at very low rate scenario. The other method developed by group member Andreas Zmija, Charge integration is working well at lower rate , and expected to be working well even at higher rate. Individual photon identification by algorithm is especially useful while performing co-relation analysis. And Neural Networks has been proven as optimal tool for solving problems, due to their ability to deal with extreme complex scenario, we

here use them do calculate photon rate for laboratory experiment as well as for photon rate calculation or star(Sirius), with different filters.

A computer would deserve to be called intelligent if it could deceive a human into believing that it was human.

Alan Turing

2 DEEP LEARNING

In this chapter, basic principles of Deep Learning are explained along with their various types and their functionality. First, in § 2.1, Machine Learning and Deep Learning are introduced. Then in § 2.2, early development are introduced. In § 2.3, basic elements of neural network are introduced with their functional behavior, and a Back-Propagation mechanism, which makes them more accurate, is also explained. In § 2.4, various types of neural networks are introduced.

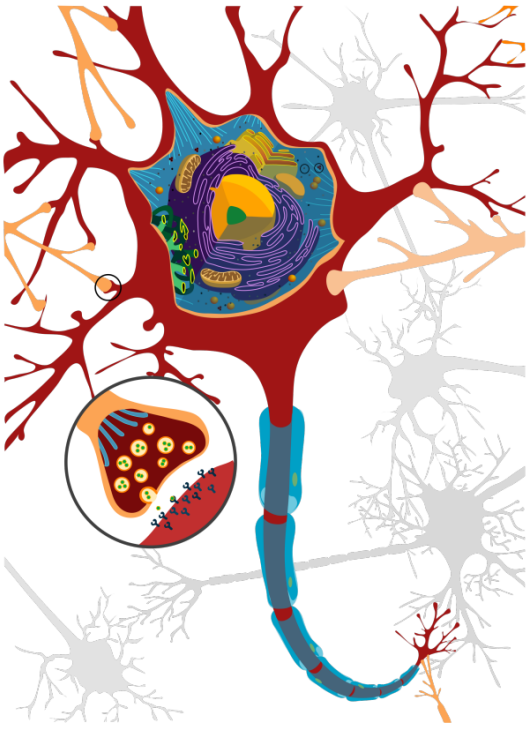
2.1 INTRODUCTION

Algorithms to analyze the pattern of data is 'Machine Learning', which requires data, and performs tasks such as classification and regression, by targeting properties of data. One can use supervised and unsupervised machine learning in this scenario. But what if we want to perform classification and regression on the data which follows a more complex pattern. In this case we require more complex assembly of machine learning algorithms, which solves problems hierarchically. This complex structure is known as Artificial Neural Networks, in which the neural network of the human brain's analogy is applied in the form for non-linear transformation on linear relation, and it's field of research is known as Deep Learning . Therefore, it is a part of machine learning, and if another technique is added which trains machines to mimic humans is known as Artificial Intelligence. For example, in the context of astrophysics, suppose you want to make your own HR-diagram of your observations and then make a nice presentation of it. You want to get your job done from your AI-robot by providing it a very large amount of different pictures. Now, it classifies the stars without human help, which is machine learning. Then from those stars it categorizes all of them according to their different properties, that is deep learning, and then it represents each star diagrammatically using all the information very carefully, that is artificial intelligence. In other example, suppose AI robot is an agent and interacts with environment(type of stars or their images), after each interaction it receives rewards of its learning. By making agent reward greedy using Q-learning and Deep Reinforcement Learning, one can achieve certain tasks. Therefore, in other words, different ML, DL and RL algorithms comes with their own ability to handle certain complexity of data. Artificial Intelligence is the field of study which consists in-depth study of all ML, DL and RL methods.

In these days a vast amount of data is available of complex patterns. Therefore, deep learning is one of the more active research fields and plays a vital role in overlap with other fields.

2.2 HISTORICAL DEVELOPMENT

Artificial neural networks are analogy of network of biological neurons, which get activated above certain threshold and pass the information. The first attempt on working of neurons was done in 1943 by neurophysiologist Warren McCulloch and mathematician Walter Pitts. They explained simple NN using model of electrical circuits. Then in 1949, Donald [12] pointed out in his book that the neural connections gets strengthened each time they are used. First simulation of hypothetical NN was done by Nathanial Rochester from IBM research laboratories in 1950[1]. In 1962, Frank, a psychologist at Cornell, was working on decision system present in the eye of fly, which determined it's flee response[13]. In order to understand it, he proposed an idea of a Perceptron, which was



Source: https://commons.wikimedia.org

Figure 2.1: Biological model of perceptron, in which Neurons are connected by synapses/dendrites. It has two states excitatory and inhibitory, depending on threshold. If neuron in perceptron is excited above threshold then it transmits the information through synapses to connected neighbour neurons. It's response behaviour is "All-or-none", means higher response is not yielded by higher stimulus. Rosenblatt has developed artificial perceprton based on this idea [13]

1 https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html

modeled on a McCulloch-Pitts neuron. He named it as Mark I Perceptron. Bernard Widrow and Marcian Hoff of Stanford developed algorithms ADALINE[14] and MADALINE[15] in 1959[16], from which ADALINE is used to predict binary sequence and MADALINE is the first neural network which is applied in order to remove echoes from telephonic conversation and is still in commercial use[2]. ADALINE basically is single layer neural network architecture with trainable weight parameters and bias, while MADALINE consists multiple ADALINE units in parallel with single neuron at the end. In both architecture, output neuron provides +1 or -1 at the end. Thus, both have functionality of AND rule i.e, if both input are True then output would be True.

Until the 1960s and 1970s, NNs were just simple connections of neurons which pass the information by getting activated, and depending on the problem and the NN connection, the computing process was done in different stages. Also during this time for supervised learning, efficient gradient descent method was used and backpropagation(BP) was developed in order to achieve optimization in training in both SL and UL[3], which was found very difficult in practice in 1980 but applied first time in 1981, and by the 1990 it became an explicit research subject. BP works on simple concept of chain rule[17] applied on parameter space with iterations which results Steepest descent[18]. Also, in case of time series prediction and sequence classification, BP plays vital role[19]. In 1979, Kunihiko Fukushima proposed an idea of multi-layer hierarchical neural network for pattern recognition and handwritten digit recognition, which was known as Neocognitron NN and became inspiration for CNNs later on[20]. This idea was originated in 1959, Hubel and Wiesel found two types of cells in visual cortex: simple cell and complex cell, which helped to extract-out features[21]. Neocogintron NN has special features such as weight pattern replication and subsampling mechanism, which are common features in some of the modern Fully Connected Neural Networkss(FCNNs). Later on, in 1992, Weng et al.[22], developed Cresceptron model which was inspired by Neocognitron and adapted topology of data during training using Max-pooling layers. Max-pooling(MP) layers reduce the size of 2D/1D arrays. Resultant each array represents a maximally active unit when downsampling is done. Later on, it's more complex version is known as "blurring" which improved object's location tolerance[23]. When MP anc CNN layers are used along side alternatively, it creates Cresceptron[24] like MPCNN, but it is trained on BP unlike Cresceptron. Doing this has certain benefits[25] such as graphic-cards has speed-up their calculations[4].

Problems related to visual attention on features can be solved by CNN, though visual pattern of data can also be seen as relation between individual data points with other data points. In other words, specific patterns presents peculiar relation between individual parts of entire visual pattern. In case of visual patter, temporal context is also important in order to understand entire pattern, for example audio, speech recognition. Use of simple neural networks, in this case, requires entire sequence to be used as an input, which is

2 https://towardsdatascience.com/a-concise-history-of-neural-networks-2070655d3fec

3 Supervised Learning(SL) consists training data with desired features as target while in Unsupervised Learning(UL) is used when target data are unavailable.

4 Flexible, High Performance Convolutional Neural Networks for Image Classification, Dan C. Cires¸an, Ueli Meier, Jonathan Masci, Luca M. Gambardella, J¨urgen Schmidhuber(2003)

unfavourable for case for time consumption and efficient calculation, because often computation are likely to suffer with inefficient memory use, difficulty in training. Breaking of sequence, and feeding them in the form of bunches causes problem of difference between spatial and temporal dimensions. Plus, in natural language processing(NLP), order would not be maintained in translation. Therefore, to overcome these problems, better approach is required, which has sequential learning behaviour,and idea comes from **Recurrent Neural Networks(RNNs)**. Origin of this approach leads to 1970s and 1980s, when J.J.Hopfield introduced the idea of physical systems with collective computational abilities[5]. The work targeted to obtain such a model which had content addressable memory for specific phase space flow of state of a physical system, with ability of parallel processing, better generalization of problem and retention of sequence via self error correction. Another important contribution, by J.L.Elman and Jordan [6], which involves study of implicit representation of the sequence rather than explicit(spatial), where providing recurrent link with state to the network enables contextual importance, where states acts as dynamic memory of neurons. In other words, use of recurrent links allows neurons to see previous outputs by providing memory to learn internal relation of time series.

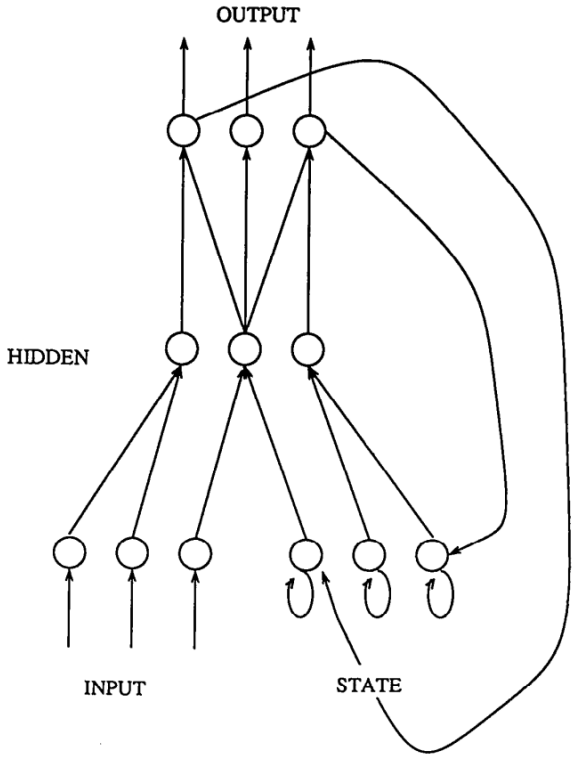


Figure 2.2: Elman neural network, where input sequence is fed to network and temporal representations are stored in the form of state, which are used for generalization of the sequence

Thus, RNN has loops which provide memory to the network and increase experience of the network, in order to understand sequential relationships by providing continuous predictions as data are fed with real-time. But simple RNN networks suffered from not recognizing the long term dependencies and vanishing gradient problem. To overcome this

5 https://www.pnas.org/content/pnas/79/8/2554.full.pdf
6 https://crl.ucsd.edu/~elman/Papers/fsit.pdf

problem in 1997 Hochreiter and Schmidhuber introduced more advanced type of neural network by introducing additional state, which was known as **Long-Short Term Memory Unites(LSTM)**[26, 27]. Design of LSTM network contains gated mechanism, which controls writing and accessing "memory" in cell state. Use of LSTM provides great benefit but many of the parameters in it are difficult to train. Thus, in 2014, K.Cho introduced modified version of LSTM, but with simpler architecture and fewer parameters[7].

In the following sections different types of neural networks and working is explained.

2.3 ELEMENTS OF NEURAL NETWORK

Neural networks are made up of neurons, and these neurons are arranged in such a way that they create layer like structure, in which each neurons has connection with previous layer neurons and next layer neurons. These neurons have characteristic of computation of input information and outputting result of these computations. Each computation includes two extra parameters, known as weight and bias, which determine the strength of connection with previous or next layer. Computation involving both weight and bias is given as,

$$y = w \cdot x + b \tag{2.1}$$

Later on, activation function is applied on this computation, which are simply non-linear functions, and determines threshold for neuron activation. If neuron is activated above certain threshold then information would be passed from neuron according to non-linearity of activation. These activation functions help neural networks to learn complex patterns in the data. And selection of activation function depends on problem. In general, computation for neuron can be given by universal approximation theorem. For entire later, generalized function for computation cam be given as,

$$F(x) = \sum_{i=1}^{N} \Phi(W_i^T X + b) \tag{2.2}$$

where, W_i^T is weight vector, and b_i is bias and $\Phi(x)$ is activation function Weight is used to determine the importance of input, means how fast activation function will trigger, whereas bias is used to delay the trigger of the activation. In this way, weight matrix is multiplied with input along with addition of bias and activation is applied. Thus, fist weight matrix is used for first layer, and out from first layer is used as input for next layer with which again new weight matrix is initialized and used in universal approximation of second layer. Similarly for deeper network, one-by-one layer use their respective initialized weight matrix for the calculation of feature importance. Below are some examples of activation functions

7 Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, et al. “Empirical evaluation of gated recurrent neural networks on sequence modeling”. In: arXiv preprint arXiv:1412.3555 (2014).

2.3.1 *Activation functions*

2.3.1.1 *Sigmoid Activation*

Sigmoid activation function is also known as logistic function, and it was the default activation function during 1990s for neural networks. Sigmoid activation function is the function used in the logistic regression. It It has ability to take any real value at input and at output it gives values in the range 0 to 1. For larger +ve value it gives output closer to 1 and for smaller value it gives output closer to 0[28]. Mathematically it can be written as,

$$R(z) = \frac{1}{1+e^{-z}} = \frac{e^x}{e^x+1} = 1 - R(-z) \tag{2.3}$$

We can find slope of $R(z)$ at any two points since the function is differential. It's use can cause vanishing gradient problem during training of neural network model at any time.

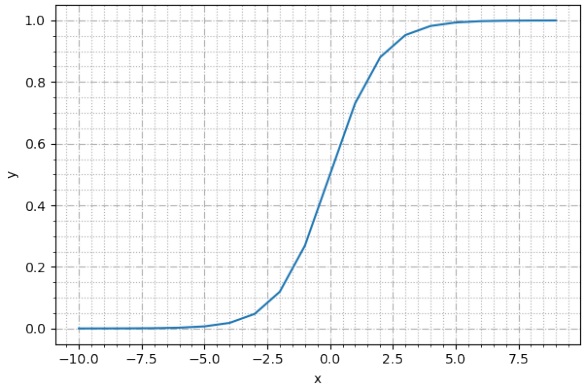


Figure 2.3: Sigmoid activation function

2.3.1.2 *Hyperbolic tangent(TanH) Activation*

Since it makes training of neural network easier and better predictive models, it was preferred over sigmoid during 1990s through 2000s. TanH activation function has ability to take any real value at input and at output it gives values in the range -1 to 1. For larger +ve value it gives output closer to 1 and for smaller value it gives output closer to -1[28]. Mathematically it can be written as,

$$R(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.4}$$

2.3.1.3 *Rectified linear unit Activation(ReLU)*

Both functions described above were well-known. Although, due to their saturation and limited sensitivity, they make neurons less informative. They makes algorithm training saturated after which it becomes challenging for it to improvise the parameter space. For

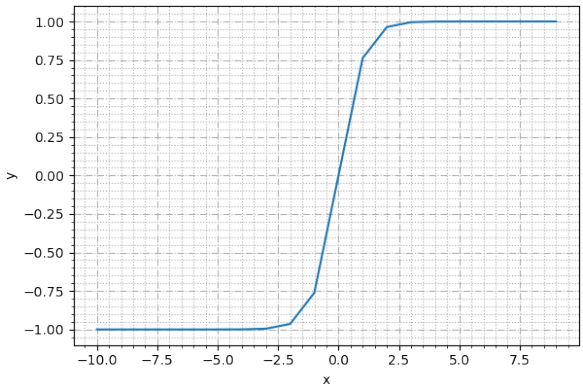


Figure 2.4: TanH activation function

Algorithm 2.1 How ReLU works.

Require: Unsorted dataset d, length of dataset n
1: **if** $input > d_i$ **then**
2: **Return** d_i
3: **else**
Return 0
4: **end if**

sigmoid and TamH, both are strongly sensitive near 0, but for higher values at -ve and +ve end, they brings algorithm in saturation phase[28].

ReLU activation function is very easy to implement and it has ability to overcome limitations of other activation functions such as Sigmoid and Tahh. Vanishing gradient is the problem in training of deep learning models which prevents models to learn features of input data, and ReLU function is less susceptible towards this problem . Therefore, nowadays it is commonly used activation function for deep learning models[28]. Mathematical definition, pseudo-code(Algorithm 2.1[8]) and exemplary image of ReLU function are given below,

$$R(z) = max(0, z) \tag{2.5}$$

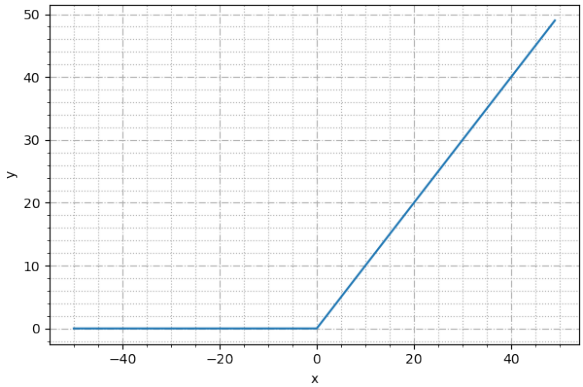


Figure 2.5: Rectified linear unit activation function(ReLU)

8 In this chapter, all the algorithms mentioned are from book[28]

Now that we know about initialization of parameter space and concept of computation in layer along with implementation of activation function, next important functions are loss function and optimizers, which plays vital role in determining how good our network is and second one is for optimisation of weights.

Since for training of neural networks is informative task, one needs to have training and target dataset. Training data is the one whose features network has to learn, while target data is considered as ground truth .i.e, true value or features from training dataset. After initializing weights network has some ability to predict solution to the problem, in other words, network is has some ability for prediction. At this point one needs to justify how close this prediction is from ground truth. There loss function is helpful to determine that.

2.3.2 *Loss function*

Loss function is the function which compares by mapping an output values of model with truth values, which is further used for optimisation of model parameters. In general, one might need to maximize or minimize the loss function according to nature of function used. Once the loss function is calculated, the closeness of the NN predictions must be estimated, here helps the maximum likelihood function. Advantage of maximum likelihood is that with the number of training examples the model prediction improves, and this property is known as "Consistency". Though, this is helpful for the case of NNs till certain point, after that all what depends is the selection of model architecture, loss optimizers and hyperparameter tuning. Selection of loss function is related to activation function of last layer. Below are few examples for loss functions. More generally, loss function is conditional log-likelihood for NN, means negative log-likelihood.

$$L_{NLL}(f_\theta(x), y) = -logP_\theta(Y = y|X = x) \tag{2.6}$$

For example, if NN has Gaussian distribution as output, with mean $f(X)$ and variance σ^2, then:

$$-logP_\theta(Y|X) = \frac{1}{2}\frac{(f(X) - Y^2)}{\sigma^2} + log(2\pi\sigma^2) \tag{2.7}$$

2.3.2.1 *Mean Absolute Error(MAE)*

It is the average of absolute difference between model prediction and ground truth. Since, it do not use squared term, it is more robust with data outliers. Mathematically it can be given as follows:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i| \tag{2.8}$$

where Y_i is true value and $\hat{Y}_i$ is prediction for n number of input data samples.

2.3.2.2 *Mean Squared Error(MSE)*

It is most commonly used for regression tasks, and can be defined as average of square of difference between model prediction and ground truth. As it has squared term, it always gives +ve output, regardless of direction. This also results model to have larger mistakes. MSE is suitable for mathematical operations since it is differential, unlike MAE.

$$MSE = E[||Y - \hat{Y}||^2] = E[||Y - f(X)||^2] = \frac{1}{n}\sum_{i=1}^{n} |Y_i - \hat{Y}_i|^2 \tag{2.9}$$

where expectation is considered over the range of training dataset, in order to gain generalization error. Minimization of the equation Eqn. 2.9 gives output's conditional expectation for given input variable X,

$$argmin E[||Y - f(X)||^2] = E[Y|X] \tag{2.10}$$

This minimization represents what neural network is trying to learn. If we replace MSE by MAE, then NN would result conditional median instead of conditional expectation. So far, we have considered mean squared error as loss function. But for the problem like classification problem where Y is in the form of discrete labels, there are loss functions which are better than MSE. For example, for Bernoulli negative log-likelihood, which gives best result for binary-classification($Y \in 0, 1$) Below are different cases of using loss function with last layer activation function.

1. **Regression problem:** real value quantity prediction
   - *Output Activation :* Linear or ReLU
   - *Loss function :* MSE, MAE or RMSE
2. **Binary classification:** one or two classes
   - *Output Activation :* Sigmoid
   - *Loss function :* Cross-entropy
3. **Multi-class Classification:** More than two classes
   - *Output Activation :* Softmax
   - *Loss function :* Cross-entropy

So far we know that NNs have parameter space and at the output loss function(L) calculate difference between prediction and true value, and at the end we use training criteria or cost $C(\theta)$(see § 2.3.6). This whole unidirectional forward process is known as **forward pass**. For algorithmic insight see Algorithm 2.2, which describes calculation of forwardpass for N layered MLP, with weight matrices θ, loss function L, cost C, regularizer ω, vector input to layer j, activation A, input vector $X = j_0$, prediction $\hat{Y}$, truth Y and bias b.

Algorithm 2.2 Forwardpass calculation

Require: Input vector X, Truth Y, Loss function L, Regularizer ω
Require: MLP with N layers, weight initialization θ, activation A
1: $j_0 = X$
2: **for** $i = 1 \dots N$ **do**
3: $\quad \boldsymbol{\alpha}^{(i)} = \boldsymbol{\theta}^{(i)} \cdot \boldsymbol{j}^{(i-1)} + \boldsymbol{b}^{(i)}$
4: $\quad \boldsymbol{j}^{(i)} = A(\boldsymbol{\alpha}^{(i)})$
5: **end for**
6: $\hat{\boldsymbol{Y}} = \boldsymbol{j}^{(N)}$
7: $\boldsymbol{C} = \boldsymbol{L}(\hat{\boldsymbol{Y}}, \boldsymbol{Y}) + \omega$

2.3.3 *Back-Propagation Algorithm*

The improvisation of prediction requires optimisation of parameters, which is possible by use of Back-propagation algorithm(BP). It simply calculates partial derivatives of C with respect to θ and determines required updates in θ for minimization of C. Since intermediate quantities such as activation(output of nodes) has influence on final prediction. Thus, one needs to consider them in recursive calculation of partial derivatives of C with respect to θ. This is the basic idea of BP. Central idea for BP algorithm is **Chain Rule**, which applies as follow for NN: small change in θ results change in intermediate quantity $f(\theta)$ by multiplying $\frac{\partial f(\theta)}{\partial \theta}$, and small change in $f(\theta)$ results change in cost function/training criteria $C(f(\theta))$ by multiplying with $\nabla_{f(\theta)} C(f(\theta))$(for computational process see Algorithm 2.3). Mathematically,

$$\nabla_\theta C(f(\theta)) = \nabla_{f(\theta)} C(f(\theta)) \cdot \frac{\partial f(\theta)}{\partial \theta} \tag{2.11}$$

where, θ is parameter weight, $f(\theta)$ is output of the node, C is cost function, and all of these are scalar quantities. If f is vector then,

$$\nabla_\theta C(f(\theta)) = \sum_i \frac{\partial C(f(\theta)}{\partial f_i(\theta)} \cdot \frac{\partial f_i(\theta)}{\partial \theta} \tag{2.12}$$

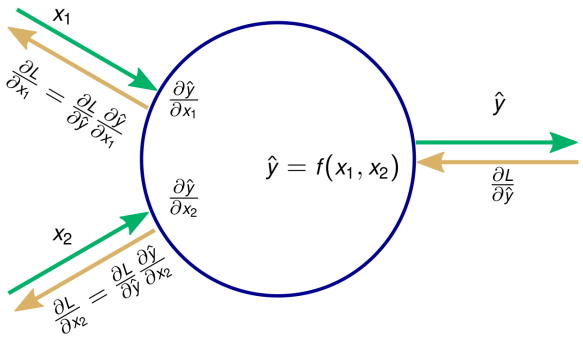


Figure 2.6: Chain rule for backpropagation algorithm for one neuron: X_1 and X_2 are input, $\hat{Y}$ is output and L is cost function[29]

Algorithm 2.3 Backwardpass calculation

Require: Input vector X, Truth Y, Loss function L, Regularizer ω
Require: MLP with N layers, weight initialization θ, activation A
After finishing forwardpass, gradient of output layer is calculated
$f \longleftarrow \nabla_{\hat{Y}} L(\boldsymbol{Y}, Y) + \nabla_{\hat{Y}} \omega$
Here ω is function of parameters, thus, yeilds 0
for $i = N \ldots 1$ **do**
    Now calculate gradient with respect to weights(parameters)
    $f \longleftarrow \nabla_{A^{(i)}} C = g \odot f'(A^{(i)})$
    $\nabla_{b^{(i)}} C = f + \nabla_{b^{(i)}} \omega$
    $\nabla_{\theta^{(i)}} C = f j^{(i-1)T} + \nabla_{\theta^{(i)}} \omega$
    Now propagate the gradients to next hidden layer
    $f \longleftarrow \nabla_{j^{(i-1)}} C = \theta^{(i)T} \cdot \nabla_{A^{(i)}} C$
end for

2.3.4 *Optimization*

With the help of BP algorithm NNs calculate the required gradient for weight updates, but do not specify the way of their use in order to implement update of weights. Optimizers fullfils this requirement. There are various gradient based learning-algorithms.

2.3.4.1 *Gradient Descent*

It is the most simplest gradient-based algorithm for NN model training. It considers small step in the direction of loss function and update the model parameters θ, and with regularizer term. Mathematically for data pair at time t $[x^t, y^t]$,

$$\theta \Longleftarrow \theta + lr \cdot \nabla_\theta \sum_t L(f(x^t; \theta), y^t; \theta) \tag{2.13}$$

where lr is step size(leraning rate), which controls the step size to take in the direction of minimization of loss function. As discussed in § 2.3.6, we can replace expectation with average training loss,

$$expectedloss = \frac{1}{n} \sum_{t=1}^{n} L(f(x^t, y^t)) \tag{2.14}$$

which means gradient of expected loss also consists of an average of the gradient obtained from each data point, which is simply an estimator of gradient with certain mean and variance. For large number of training examples with batchwise training, known as batch gradient descent, which has relatively small variance. While in gradient descent, each step involves gradient computations for all training examples, this method alone is inefficient. Sometimes GD makes learning process slow, specially when gradient is very small. In order to overcome these problems, Momentum method is designed to accelerate learning, specially when gradients are small[30]. Assume we have a solid sphere on the horizontal surface. Now if the gentle slope is applied to the surface, the ball starts moving in that direction with increasing speed over time. Likewise momentum method is designed on

Algorithm 2.4 SDG with momentum

Require: Learning rate lr, momentum parameter α
Require: Initialized weight parameter θ, initial velocity v
1: **while** Criteria for stopping not met **do**
2: get minibatch of n samples from training dataset
3: set $f = 0$
4: **for** $t = 1 \dots n$ **do**
5: Calculate gradient: $f \longleftarrow f + \nabla_\theta L(f(x^t;\theta), y^t)$
6: **end for**
7: velocity update : $v \longleftarrow \alpha v - lrf$
8: Update weights: $\theta \longleftarrow \theta + v$
9: **end while**

this principle. The update rule remains of the same form with small change of inclusion of variable$v$, which plays role of velocity and α is momentum:

$$v \longleftarrow \alpha v + lr\nabla_\theta \frac{1}{n} \sum_{t=1}^{n} L(f(x^t;\theta), y^t) \tag{2.15}$$

with $\theta \longleftarrow \theta + v$.

2.3.5 *Adam optimizer*

Adam was first introduced in ICLR(International Conference on Learning Representations) 2015[31], with showing impressive performance gains in terms of training speed, however in some cases SGD provides better results. It's algorithm uses the adaptive learning rate and determines learning rates for each parameters. It has properties of both Adagrad[32] and RMSprop. Thus, like Adagrad it works well for sparse gradient settings but troubling with non-convex optimization, but it has some of the properties of RMSprop which resolve some of Adagrad problems. According to article by Andrej Karpathy[9], Adam is becoming more and more popular. Adam also have property of GD with momentum. Like RMSprop, it calculates squared gradients in order to scale learning rate, while it uses moving average of gradient like SGD.

Adam optimizer is designed for optimization process of neural networks, and since in order to adapt learning rate for each weight of neural network, it uses estimation fo first and second moments(expected value of a variable) of gradient, it is known as Adam(Adaptive moment estimation). The n^{th} moment of a variable can be given by,

$$m_n = E[X^n] \tag{2.16}$$

where, m is moment and X is some variable. One can compare this with neural network cost function acting as gradient, for each mini-batch of data. Mean is the first moment and second moment is uncentered variance, and we also know that after each mini-batch the

9 https://karpathy.medium.com/a-peek-at-trends-in-machine-learning-ab8a1085a106

gradient is calculated. Adam use both of the mentioned information above and calculates moving averages for each mini-batch.

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \end{aligned} \tag{2.17}$$

where, m and v are moving averages and g is current mini-batch gradient, β_1, β_2(0.9,0.999) new introduced hyperparameters. Since both m and v are estimates of first and second moments, we expect that following relation should be satisfied for unbiased estimator.

$$\begin{aligned} E[m_t] &= E[g_t] \\ E[v_t] &= E[g_t^2] \end{aligned} \tag{2.18}$$

But for moving average this do not hold true, as Adam initializes with 0 moments. For example,

$$\begin{aligned} m_0 &= 0 \\ m_1 &= \beta_1 m_0 + (1-\beta_1) g_1 = (1-\beta_1) g_1 \\ m_2 &= \beta_1 m_1 + (1-\beta_1) g_2 = \beta_1 (1-\beta_1) g_1 + (1-\beta_1) g_2 \\ m_3 &= \beta_1 m_2 + (1-\beta_1) g_3 = \beta_1^2 (1-\beta_1) g_1 + \beta_1 (1-\beta_1) g_2 + (1-\beta_1) g_3 \end{aligned} \tag{2.19}$$

from Eqn. 2.19 it is clear that further expansion of ms, the contribution of first gradient term decreases as they get multiplied by smaller βs. Here example is given only for first moment but one can similarly obtain second moments. Rewriting Eqn. 2.19 for moving average,

$$m_t = (1-\beta_1) \sum_{i=0}^{t} \beta_1^{t-i} g_i \tag{2.20}$$

Now take expectation at both side in order to check how it relates to the true first moment.

$$\begin{aligned} E[m_t] &= E[(1-\beta_1) \sum_{i=1}^{t} \beta_1^{t-i} g_i] \\ &= E[g_i](1-\beta_1) \sum_{i=1}^{t} \beta_1^{t-i} + \zeta \end{aligned} \tag{2.21}$$

where ζ is error. Now that we have estimators, which are biased, we one need to apply bias correction as follow:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \end{aligned} \tag{2.22}$$

Now, using moving averages one can obtain scaled learning rate for for each parameter, and weight are updated using following relation:

$$\theta_t = \theta_{t-1} - lr \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} \tag{2.23}$$

Algorithm 2.5 Adam optimizer

Require: Learning rate:lr, Decay rates:$\beta_1, \beta_2 \in [0, 1)$
Require: Initialized weight parameter θ_0
$m_0 \longleftarrow 0$ (Initialize 1^{st} moment vector)
$v_0 \longleftarrow 0$ (Initialize 2^{nd} moment vector)
$t \longleftarrow 0$ (Time-step initialization)
while θ_t not converged **do**
  $t \longleftarrow t + 1$
  $g_t \longleftarrow \nabla_\theta f_t(\theta_{t-1})$: Get gradients
  $m_t \longleftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$: First moment estimation(biased)
  $v_t \longleftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) g_t^2$: Second moment estimation(biased)
  $\hat{m}_t \longleftarrow \frac{m_t}{(1-\beta_1^t)}$(First moment: bias correction)
  $\hat{v}_t \longleftarrow \frac{v_t}{(1-\beta_2^t)}$(Second moment: bias correction)
  $\theta_t \longleftarrow \theta_{t-1} - lr \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$(Apply weight update)
end while

where ϵ is the term which prevents second term to become 0. Algorithm from original paper from authors is given below(Algorithm 2.5)

In addition to that, Adam has two more plus points. One is that, for each iteration it has specific learning rate, which is simply addition of intuitive understanding to previous learning rate. And magnitude of gradient from Adam update rule is invariant, which helps dealing with very small gradients.

However, despite of better training, in some cases Adam does not converge to optimal solution. For example, for CIFAR dataset, state-of-the-art results can only be obtained by using SGD[33], while Adam optimizer only covers some of the marginal values. There are also other papers which discuss about limitations of Adam optimizer[34].

2.3.6 *Training criteria and Regularizers(L1 and L2)*

Minimization of Eqn. 2.7 gives distribution of generated data. But in practice, since we do not know $P(X, Y)$ and computation of minimization is extremely complex, it is more convenient to approximately minimize training criteria. For example, the simplest way to minimize training criteria or cost($C(\theta)$) is to average the training loss, $L = \frac{1}{n} \sum_{i=1}^{n} L(f_\theta(x^j, y^j))$ for n examples of training samples(x^j, y^j). However, there is also chance of over-fitting and under-fitting of model. In that case, regularization is useful way of making model more generalized. Regularization is combining evidence from training loss(input data) with prior values of parameter θ. The most common way of using regularization is just using extra term(regularization term), which is multiplication of regularization coefficient λ and parameter's squared norm($\theta = ||\theta||_2^2$), and this regularization is known as **L2 regularization, weight decay or Ridge regression**. It is called as "weight decay" because it put penalty on weights and updates them using regularizer term with gradient calculating al-

gorithm such as Gradient descent. Mathematical formulation of weight update with using gradient descent(learning rate ϵ) is given below,

$$\theta \longleftarrow \theta - lr \cdot \Delta_\theta L(f_\theta(x), y) - lr \cdot \lambda\theta = \theta(1 - lr \cdot \lambda) - lr \cdot \Delta_\theta L(f_\theta(x), y) \tag{2.24}$$

Another famous regularizer is **L1 regularizer** also known as **Lasso regularizer**, in which we simply use ($\theta = ||\theta||$). $L1$ regularizer keeps pushing values of weight to zero even if regularizing term becomes very small close to 0. As a consequence, value of some of the parameters becomes close to zero. On the other hand, $L2$ regularizer punishes weights aggressively and avoids large values of parameters.

2.4 TYPES OF NEURAL NETWORKS

2.4.1 *Fully-connected neural network*

Fully Connected Neural Networks are the simplest neural networks, which consist of layers of neurons working on basic principle. In it each of the neurons from one ´layer is connected to each of the neurons of other neurons., In this way it forms fully connected architecture. This proliferates number of parameters of the network, which sometimes cause overfitting.

2.4.2 *Convolutional Neural Network(CNN)*

In general, convolution is a mathematical operation applied on two functions and the result of this implementation gives third function, which determines that how shape of one function depends on other function's shape. Neural network which simply uses "Convolution" instead of matrix multiplication is known as Convolutional Neural Networks(CNN). For example, we have two functions namely, $x(\tau)$ determining position of spaceship, and $w(\tau)$ is the weights to each position according to time τ. Applying convolutional operation provides smooth estimate of spaceship position,

$$s(t) = \int x(\tau)w(\tau)d\tau \tag{2.25}$$

which can also be written as below and $'*'$ is used for convolutional operation in general.

$$s(t) = (x * w)(t) = \sum_{\tau=-\infty}^{\infty} x(\tau)w(t - \tau) \tag{2.26}$$

In equations above, first argument, x is referred as an input(an array of data), second one w is as kernel(an array of learn able parameters) and output as feature-map. Both input and kernel must be stored separately, and they provide finite number of summation array according to size of input. For the case of input data having more than one dimensions,

one can also apply convolution. FOr example, for 2D input image I, convolution can be written as,

$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[m,n]K[i-m, j-n] \tag{2.27}$$

From the property of convolution operation of being commutative, one can also rewrite,

$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[i-m, j-n]K[m,n] \tag{2.28}$$

Convolution is not actually matrix multiplication. It is useful to consider convolution as matrix multiplication but this idea is not useful for designing it's algorithm and understanding its working. This means that it is not strongly dependent on matrix multiplication, which makes convolution operation as preferable choice in many cases.

Three important ideas, sparse interactions, parameter sharing and equvarient representation, makes convolution algorithm beneficial. In addition to that, it can be also operable with variable size of input. Below all three ideas are explained in detail.

2.4.2.1 *Sparse interactions and parameter sharing*

FCNN are based on matrix multiplication, means interaction between entire input with all units. Unlike that convolution networks are based on sparse interactions, which is accomplished with the help of kernel size smaller than input size. For example, if input image is of size 224×224 with having 3 channel(R,G,B), then one can select kernel size even smaller such as 3×3×3, or larger, and having this kind of functionality provides benefit of having fewer parameters to store(means fewer computations), it also gives importance to smaller features. This improves the statistical efficiency of the process. For m inputs and n outputs, the parameters would be $m \times n$ and algorithm have $O(m \times n)$ examples. This is for the case of matrix multiplication in FCNN, but for sparsely connected NN, input size would be fxed(smaller than input) which makes number of parameters $k \times n$ and $O(k \times n)$ examples. This means from set of given number of kernels each member is used at every position of the input.

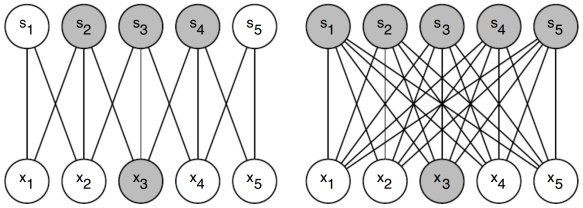


Figure 2.7: Representation of traditional neural network(right) and convolutional network(left)[28]

2.4.3 *Equivariance*

With the change in input if the output changes then one can say that the function have property of being equivariant. For example, function f and g follows the relation $f(g(x)) = g(f(x))$ then, f is equivariant to function g. This means that with the shift of kernel the

selected part from input image would also be shifted and the output would also be created with shift. This is useful to detect features and specially at edges.

If a function which is required to be learned by layer have direct connection with local transformations then use of convolution becomes dramatically more efficient, otherwise it will make model full of error.

2.4.4 *Pooling*

Typically, convolutional neural network consists of three types of stages. In first stage, numerous layer performs convolutions in parallel for generation of presynaptic activation. In second stage, non-linear activation is applied over this presynaptic output, which is known as *detector stage*. In third stage, *pooling* function is applied to modify the output and output is replaced with summary statistics of the nearby outputs. This makes the representation *invariant* for small input translations. It is better to use fewer pooling layers since it summarizes the response over a whole neighborhood

2.4.5 *Recurrent Neural Network(RNN)*

As discussed in § 2.1, Recurrent neural networks(RNN) consist of connection between nodes in such a way that connections make directed graphical pattern in temporal sequence by enabling ´learning of temporal dynamics. By using their internal memory states, they are able to process inputs with variable length. Main problem with using simple RNN is that they are only able to learn short term sequence accurately. For long enough sequence they suffer from carrying information from one time-step to another one, which is caused by vanishing gradient problem during backpropagation. In other words, since gradient update is crucial for neural network optimization, shrinking of gradient in RNN for long sequence makes trouble for weight updates by making model under-trained. As a solution for learning long-term dependencies in sequence, Long-short Term Memory(LSTM) units and Gated Recurrent Units(GRU) were introduced, which are explained in § 2.4.5.1.

2.4.5.1 *LSTMs and GRUs*

Both LSTMs and GRUs have internal mechanisms called gates, which regulates information flow in each cells of LSTM and GRU. They learn important information and throw away unnecessary information, which make long chain of sequence of cells to learn only relevant information. Basic idea of learning mechanism of LSTM and GRU is similar to RNN. First it takes input sequences and process them and store some information in memory called hidden state. It passes this stored information to next time cell. Now in next cell, information obtained from previous cell's hidden state and input information of present cell get added in the form of vector. Thus, formed vector has information of previous state and current input. Now, this vector is scaled in tanh function which is simply the memory(calculated hidden state) of present state. Benefit of tanh function is that is results

values in between -1 and +1, instead of exploding values by multiplications. This is how RNN works.

LSTM cell has the similar working principle, but with much more complicated mathematical operations, which allows LSTM to keep relevant information and forget irrelevant ones. In the heart of computations in LSTM are calculation of cell state and gates. One can consider cell state as a highway, which passes important information in entire sequence, which means information from earlier states are also used by last states in sequence, and helps in reduction of short-term memory. Through-out the sequence, information gets added into cell states by gated calculations in cells of sequence. In gates sigmoid activation is used, which scales information from 0 to 1, which is helpful to extract information to forget. Because information multiplied with closer to 1 gets more importance while information multiplied by closer to 0 gets less importance. Basically, LSTM cells have three different gates, forget gate, input gate and output gate.

Information from previous hidden state and current inputs are added and resultant vector is passed through sigmoid activation, which determines important information to forget. Now, this sigmoid-scaled output is point-wise multiplied with cell state, which adds forgetting effect of unnecessary information. This is known as 'forget gate'. Now, two copies of vector obtained from current input and previous hidden state are created, one is passed through sigmoid activation from which gives it important information, and the other is passed through tahh activation scale values from -1 to +1 which becomes helpful to regulate operation. Now, output from both activations gets multiplied and its output is used to update cell state, by simple addition, in order to add relevant information. This is known as 'input gate'. The last state is 'output gate' which determines next hidden state. One copy from addition vector(previous hidden state+current input), is passed through sigmoid activation, and on other hand one copy of cell state(after forget gate update and input gate update) is passed through tanh activation,and output of both are point-wise multiplied, which gives a vector, which is simply calculated hidden state to be passed to next cell. In short, forget gate decides what information to forget, input gate decides relevant information to be added, and output gate determines what information should be passed to next cell. This is how LSTM works.

GRU is similar to LSTM but with more simpler structure. The difference are that GRU has no cell state, and it uses hidden state for information transfer, and it has only two gates, reset gate and update gate. Update gate is combination of forget gate and input gate from LSTM, which determines important information to keep and rest to be discarded.While reset gate determines how much information from past information to forget. GRU is bit more faster than LSTM, and sometimes performing better than LSTM. It is not determined which one is winner. Therefore, researcher tries both and determines which one is better for their case.

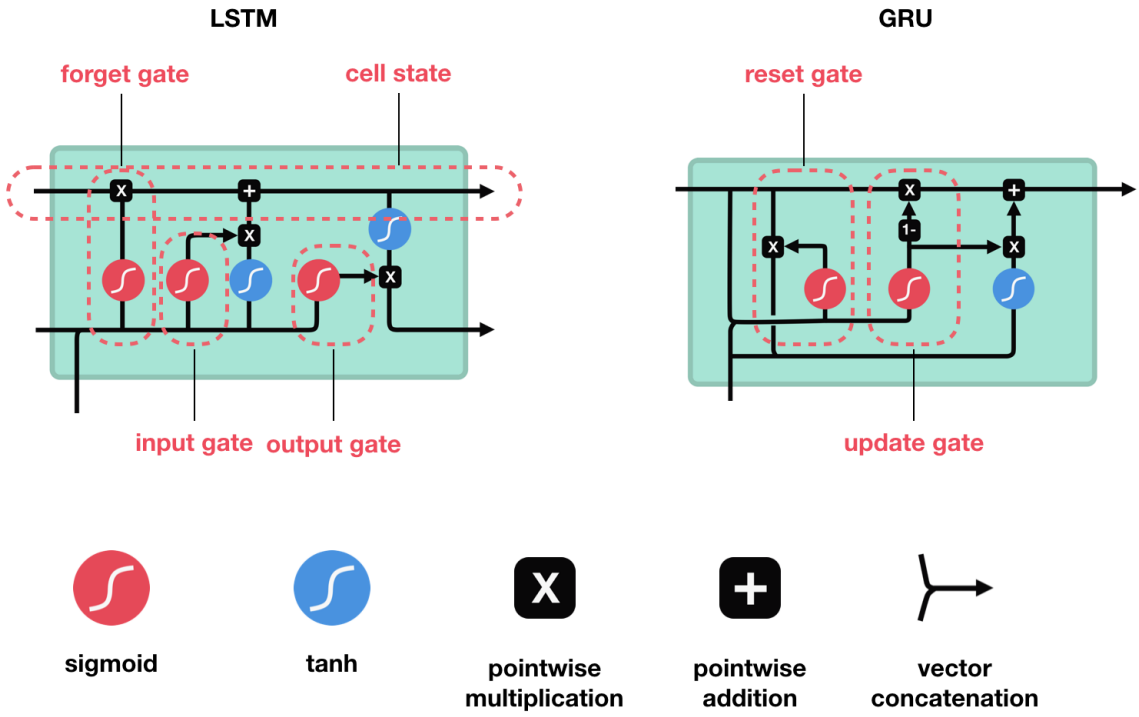


Figure 2.8: Architectures of LSTM and GRU, with different gates; LSTMs consists input gate, output gate and forget gate, while GRU has reset and update gate. All gates have different functions[35].

Artificial Intelligence is the new electricity!

Andrew Ng

3 ELECTRONIC SETUP AND DATA PREPARATION

In this chapter, first of all, instrumental setup,data generation and pre-processing are explained. Using this method, various set of data are created for neural network training. In addition to that, there are two other methods which are used to compare neural network model predictions, charge integration and peak finding method, which are also explained in this chapter. In addition to that, setup required experimentation for photon detection is also explained.

3.1 PHOTOMULTIPLIER TUBES(PMT)

For photon detection in our experimentation, we use photomultiplier tubes since they are highly sensitive and they have fast response. Photomultiplier tubes are operated on mainly two principles, photon to electron conversion via photoelectric effect and amplification. The second one indicates that when metal or semiconductor are exposed to light, they emit electrons[36]. Generally, under almost vacuum condition in tube like structure, PMTs have an entrance window from where light enters the device, This tube consists photocathod, electron multiplier and collector. PMTs detects light and provides electrical signal. This signal then amplified by secondary electron emission.

First of all, the light pass through the entrance window, then incident photons from light excite the photocathode electrons, which get emitted in vacuum, and these electrons are known as photoelectrons. Then several focusing electrodes play role of focusing and acceleration of photoelectrons towards electron multipliers. Electron multipliers consist several dynodes. For example, for the case of first dynode, after it gets hit, because of secondary emission electrons get multiplied. This process takes place at each dynode one by one. At the end all the secondary electrons are collected by anode(after last dynode), and used to provide output signal. Because of these principles PMTs provide signal with ultrafast response and high sensitivity[36, 37]. One big advantage of using PMT is that, it uses repetition of the secondary emission process, which means, even if we observe very small current at the photocathode in the beginning, there repetition process at each dynode amplifies it and at the end amplified current is obtained. This gain depends on average secondary emission(multiplication) factor δ from each dynode. For example, for n dynode PMT, the current amplification is $G = \delta^n$[37]. Another important property for PMT is "Quantum efficiency", which is probability of triggering of electrons. In other words, it is simply ratio of the number of photoelectrons from photocathode to the number of incident

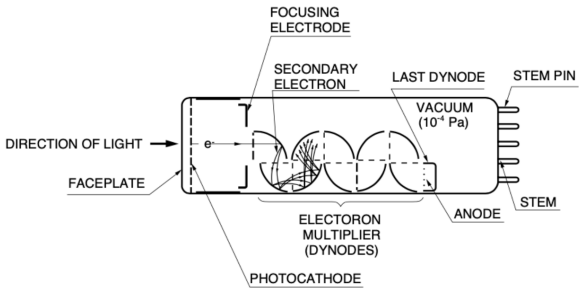


Figure 3.1: Schematic diagram of PhotoMultiplier Tubes[37]

photons. PMTs have certain probability to collect triggered photoelectron at first dynode, which is known as collection efficiency(CE). While ability to detect events is detection efficiency(DE), which is given by ration between detected number of event and incident number of photons. DE can also be given by product of QE and CE[36, 37].

3.2 AMPLIFIER

For amplification of PMT signal, TA1000B-200 is used, which is DC coupled device. It is very low noise preamplifier and doesn't affect signal except providing an offset, which can later on set to 0. For signal bandwidth of 400MHz, it is very sensitive and has very fast rising time, and leads to amplification by the factor of 200, which is very high for PMT. Thus, amplifier has been remodeled in such a way that amplification factor is 10. Once signal is amplified, it is sent to digitization card directly[38, 39].

3.3 DIGITIZATION CARD

In series of experiments for high speed data acquisition, digitization card by Spectrum M4i.2212-x8 is used. It is connected with amplifier via one of four available input channels, in order to analyze PMT signals. It is capable of sampling rates upto 1.25 GS/s with largest bandwidth upto 500MHz [40]. For graphical representation, "SBench 6" software is used. All the channels are adjustable and sampling clock can be set according to requirement. For example, voltage range of each channel can be set to $\pm$40, $\pm$100, $\pm$200, $\pm$500 mV, and sampling clock can also be set to 0.8 or 1.6 ns(for this project 1.6ns is selected). With selected input mode (2GS/s) the file with 1000,000 datapoints have duration of 160μs. In graphical presentation by software, the y-axis denotes Amplitude, in ADC counts, which later on can be converted into mV[40].

3.4 GRAPHICS PROCESSING UNIT

To speed up neural network calculations in all experiments Graphic Processing unit(GPU) form NVIDIA is used. At the beginning GeForce GTX1080 was used for test runs, which has 6 GB RAM. While later on all the neural networks were run over more advanced GPU, GeForceRTX 2080Ti with 4352 cuda cores with 12GB of RAM. Use of GPU speed

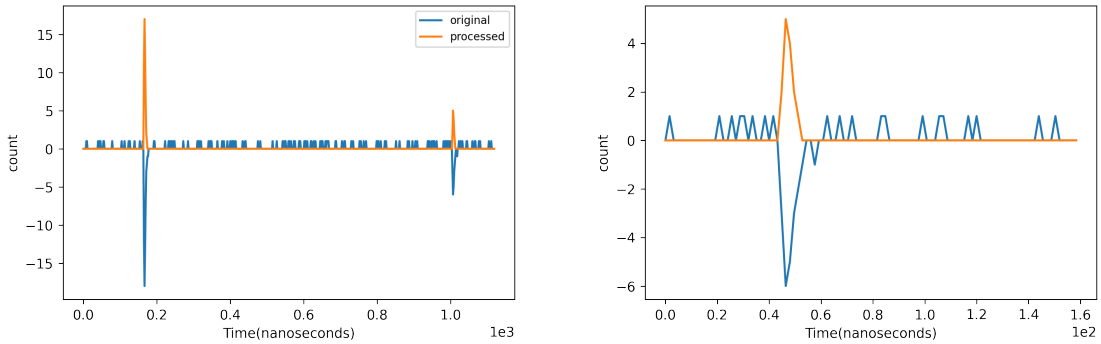


Figure 3.2: Raw waveform obtained from experiment

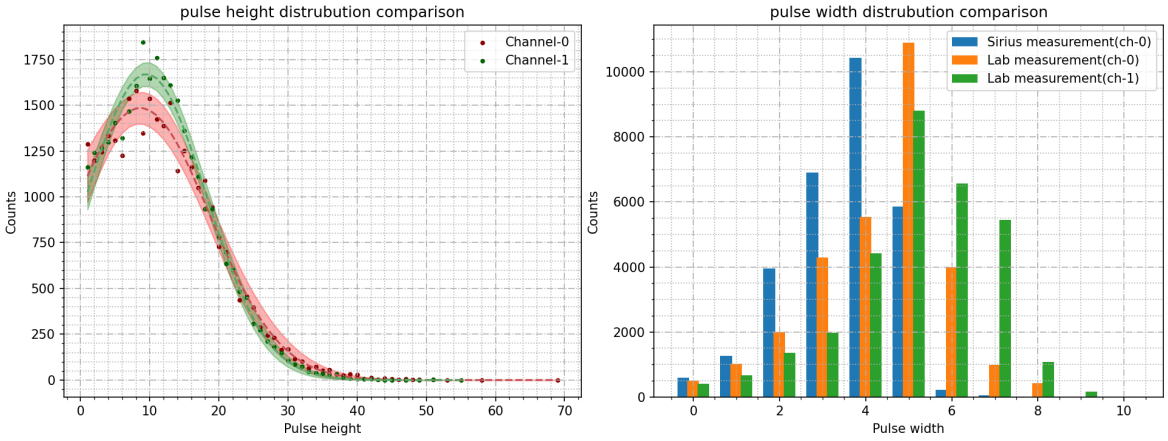
up the calculation process but it is not used for data preprocessing and creation of datasets. "Tensorflow" is used for neural network model creation and special settings were established so that Tensorflow can use GPU for calculations.

This thesis covers two type of experiments for rate calculation, one is lab experiment. For lab measurements the set of PMTs used is same as it was used in experiments performed by Naomi Vogel[41]. Laboratory experiments were performed by Andreas Zmija with different transmittance.

3.5 DATA PREPROCESSING

In this project, data from two types of experiments are used. We also have used in some cases normalization method. First of all, experiments are performed and *.bin* files are obtained which include pulse pattern over time from experiment performed. Then these files are converted into numpy readable format.

After that, if the waveform is generated with some offset then off-set is removed. Then baseline noise is removed by applying threshold. Threshold is determined by considering height of smallest occurring spike. Then processed waveform would only have photon peaks, from which photon pulse shapes are extracted. Now we have set of samples, each consisting single photons. This set is splitted into train and test data in ratio of 8:2. Further, training set is splitted into again actual training set and validation set with ratio of 8:2. Then all the train, validation and test sets are used to generate MC dataset(Algorithm 3.1). Thus, we stick to the size of MC-dataset, and created every time 2,37,900 samples for training, 1,22,000 samples for validation and 244,000 samples for test dataset. All of these datasets have equal number of examples for each rate. For evaluation identical process is used but with control over photon bunching by considering average distance between successive photons in sample as 0,1,2,3,4,5. For example if the average distance between photon is greater than 0.5 then it goes to class of average distance between photon peaks as 1. The same method of MC data generation is used for waveforms obtained from ex-



(a) Pulse height distribution of photons in waveform (b) Pulse width distribution of photon from waveform

Figure 3.3: Data analysis of extracted photon pulses from waveform

periment performed for rate measurements from Sirius. And at the end using measured photon rate, flux can be easily calculated from following relation,

$$R = F \cdot A \cdot B \tag{3.1}$$

where, R(MHz) is measured photon rate, F($Mhz \cdot m^{-2} \cdot nm^{-1}$) is flux, A($m^2$) is area and B is band width(nm) of filter used.

3.6 CHARGE INTEGRATION

This method is originally developed by Andreas Zmija. It can also be used without preprocessing but for fair comparison we have used preprocessed data even for charge integration method. Here, steps required to perform charge integration method are mentioned, as shown in figure below. One of the requirement of this method is that offsets in the waveform is needed to be subtracted before performing any task mentioned below.

1. Take measurement with low photon rate(about 1 MHz), which provides complete photon pulses.

2. Now, considering noise, by finding all peaks, create pulse height histogram. Then fitting curve would be be Gaussian, with cut at zero, since positive peaks are not physical. From this distribution average pulse height an be calculated In the example below is -13.2.

3. Now, execute peak find algorithm from Scipy library but only on large pulse shapes and by synchronizing all the pulses overlay them at particular peak position on x-axis. After that, normalize each pulse by dividing simply it's maximum values, which will convert each pulse height to 1. Then divide each y value by the number of used pulses, which will provide average pulse shape of all the normalized pulses. And then calculate the integral of the average normalized pulse.

Algorithm 3.1 Monte-Carlo algorithm

Require: Input vector Ph, number of photons required to add in one sample
Require: Matrix of unique shaped photon samples, x

$j_0 = X$
for $Ph = 1 \dots N$ **do**
  a= set of empty samples of fixed sample size(1000×1000)
  ar= set of empty array, number of photons added into each samples of s(1000)
  for $a = 0 \dots 1000$ **do**
    $SepMatrix$= randomly selected single photon sample form x
    for $j = 1 \dots 1000(SampleSize)$ **do**
      $TimeStep$= randomly select time step where peak of the selected photon sample is required to be added
      $width$= length of non-zero part of single photon sample
      $NonZeroPart$= non-zero part of sample
      if $width + TimeStep > SampleLength(1000)$ **then**
        Slice the latter part from $NonZeroPart$, and use it for adding it into a
      else
        if $TimeStep - width < 0$ **then**
          Slice the former part from $NonZeroPart$, and use it for adding it into a
        else
          Use non-zero part for addition
        end if
      end if
    end for
  end for
  update samples of a
  update umber of photons added in ar
end for

4. Average photon charge can now be calculated by,

$$Avg.Charge = Avgpulseheight \cdot Avg.normalizepulsesum \tag{3.2}$$

5. Rate calculation: for any calculated waveform, from mentioned steps above, photon rate is simply the mean value of the waveform. This is also applicable on waveform at high rate, where photon shapes overlap with each other. Rate calculation equation is given below,

$$Rate = \frac{Waveformmean}{Avg.charge \cdot 1.6nanoseconds} \tag{3.3}$$

where, 1.6 nanoseconds is the time sampling width.

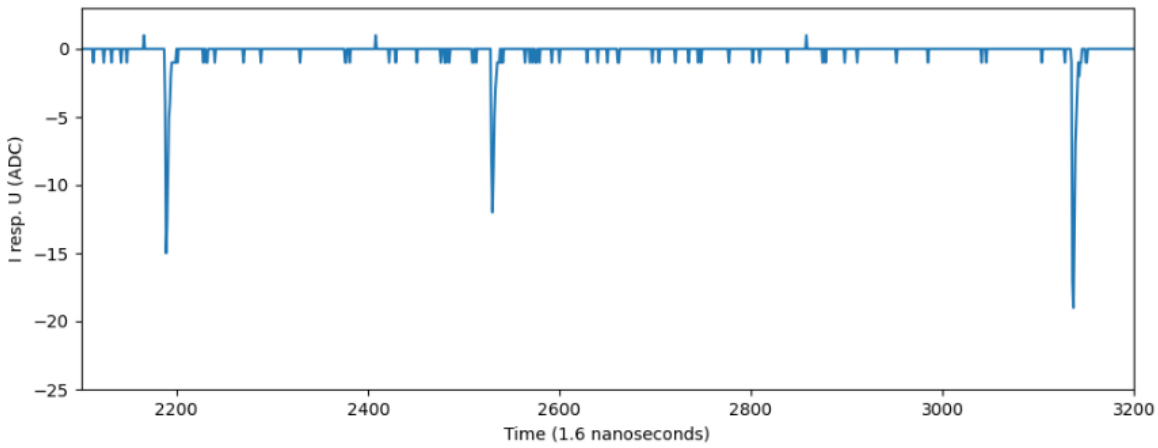
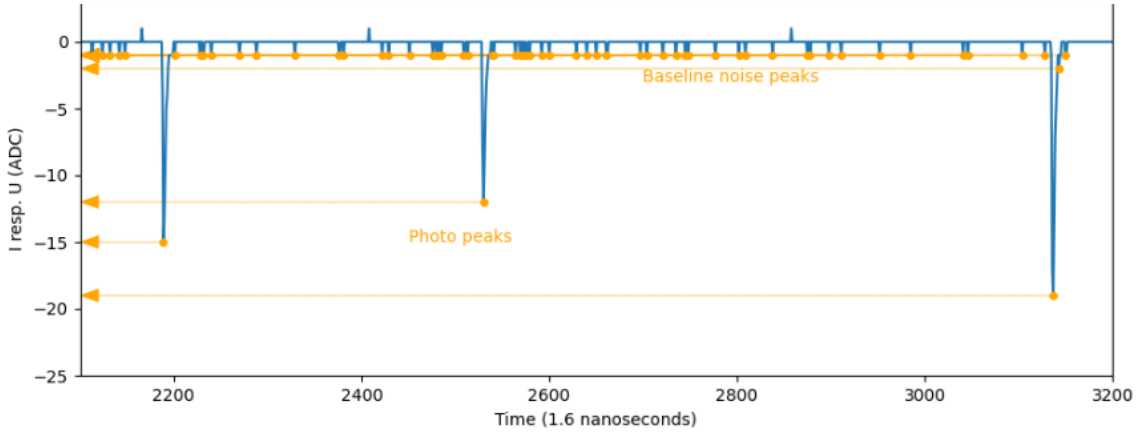
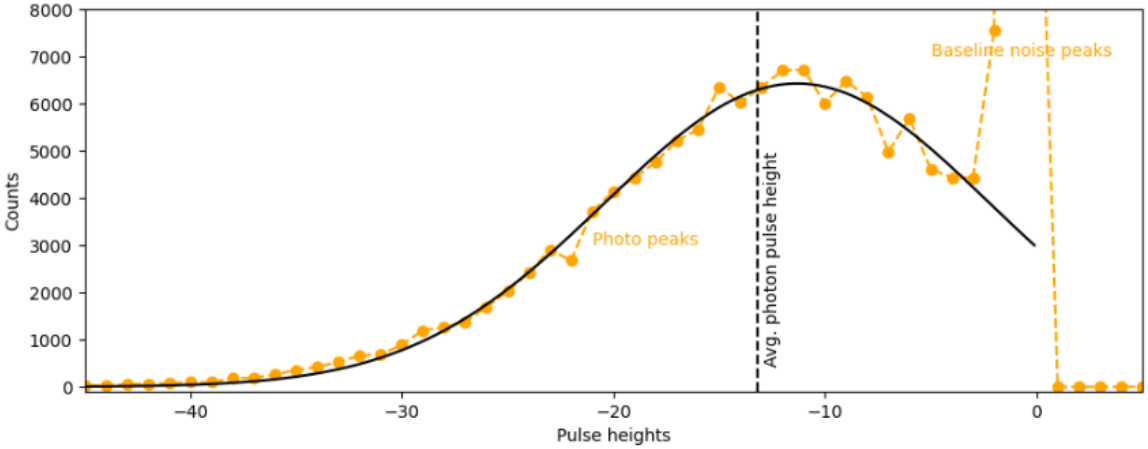


Figure 3.4: Calibration

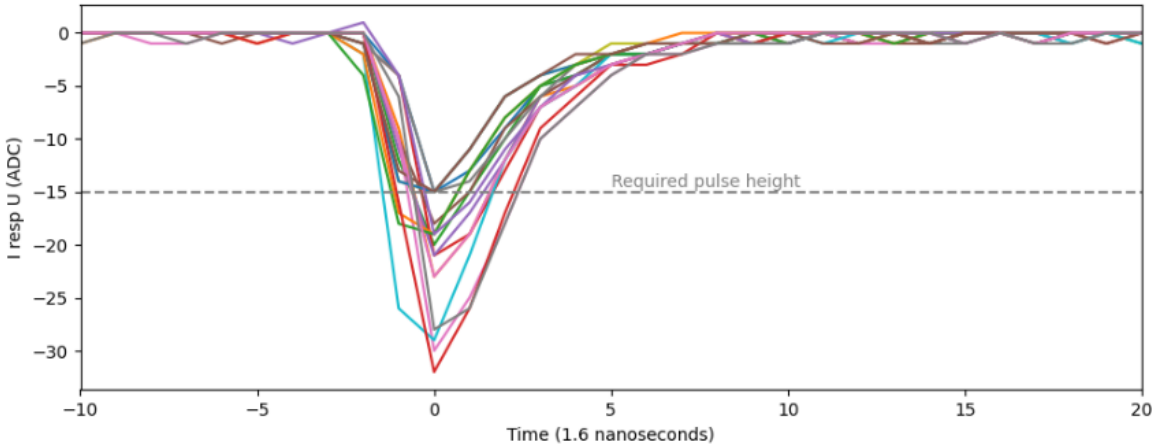


(a) Pulse height determination

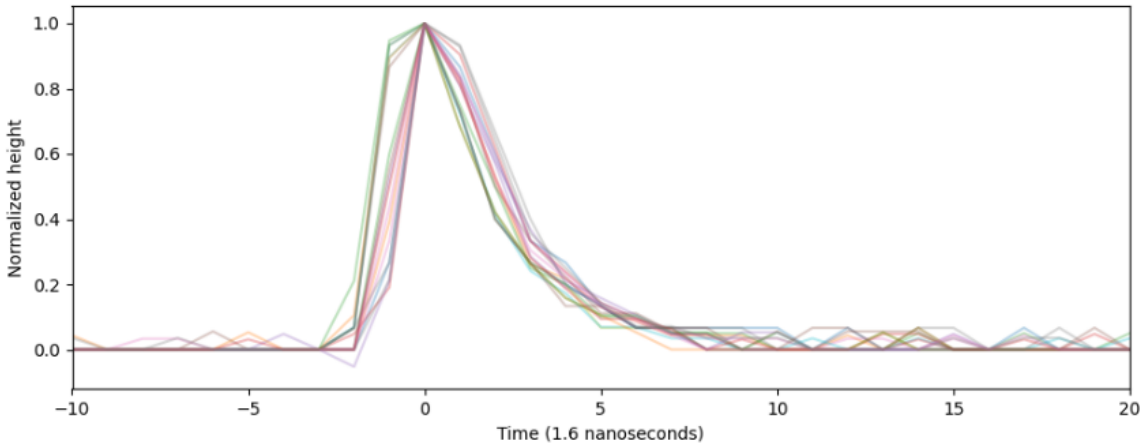


(b) Pulse height histogram

Figure 3.5: Average photon pulse height

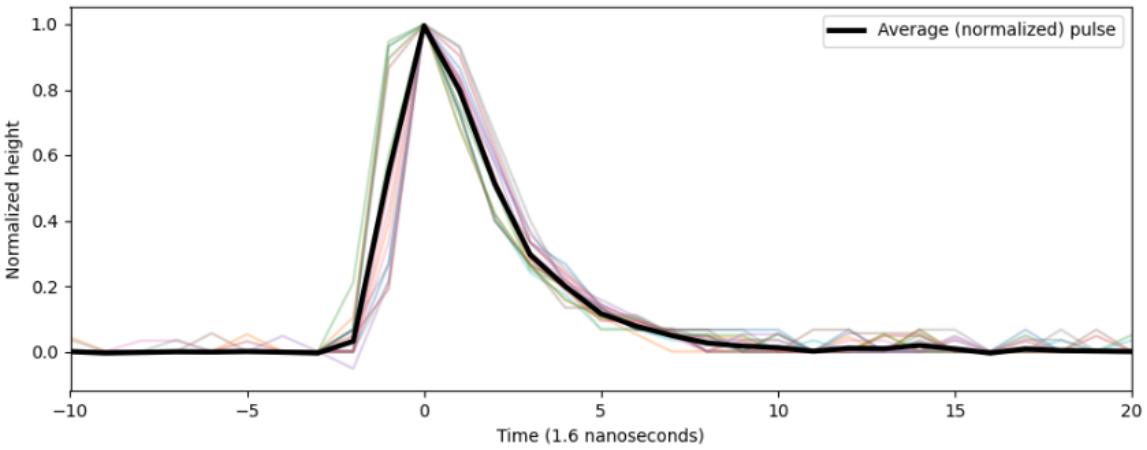


(a) Ppeak find implementation

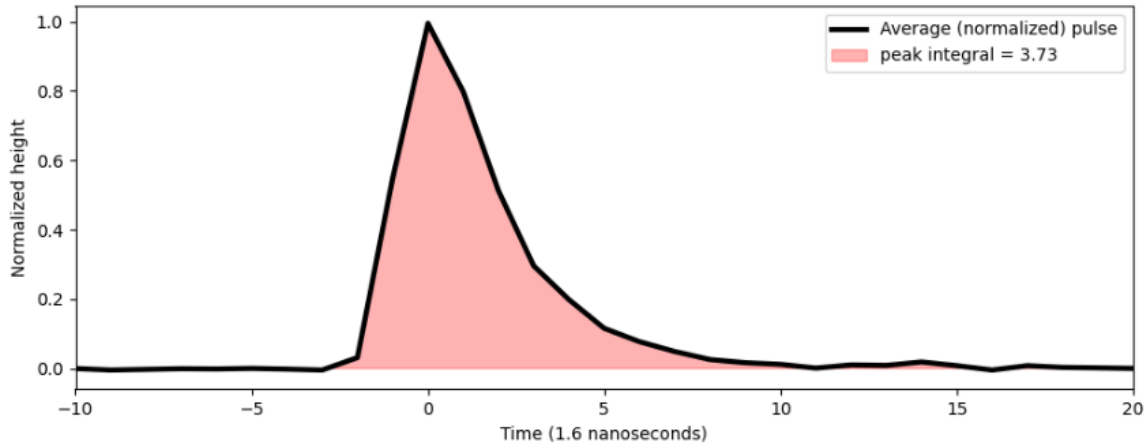


(b) Normalization

Figure 3.6: Average photon pulse shape



(a) Average pulse shape



(b) Pulse shape integral calculation

Figure 3.7: Pulse shape integral

Our intelligence is what makes us human, and AI is an extension of that quality.

Yann LeCun

4 RESULTS

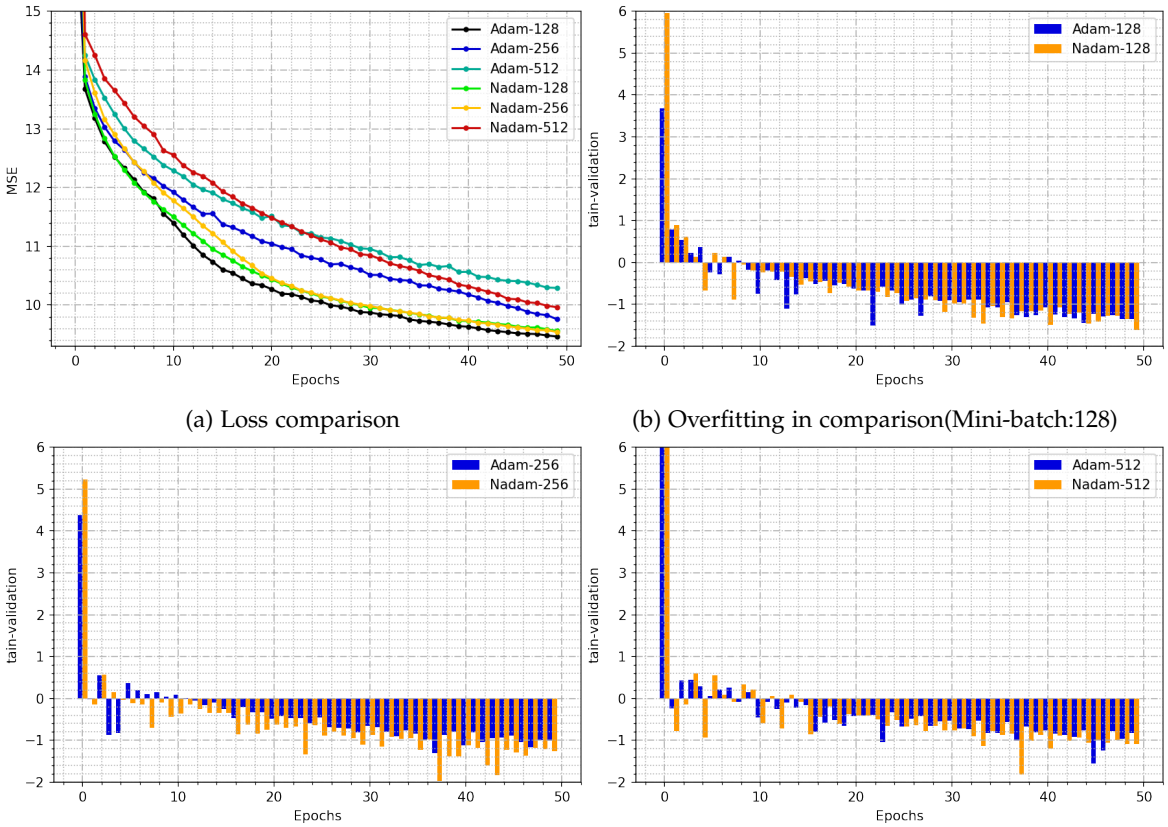
In this chapter, first of all step-by-step description is provided for all attempts. Training of different types of network and approach is explained in detail in this chapter. After training the model evaluation step is performed, and then network predicitions are compared with two different methods, Charge Integration developed by Andreas Zmija and Peak find method developed by Scipy library.

First of all it is important to determine for which configuration of optimizer, loss function and learning rate with momentum network provides optimum results. And to determine that one needs to simply look at loss curve, where loss represents learning of model. Here, by model, one can refer NN architecture. Here, for simplest model architecture(MLP), we decided three layer model architecture and trained for different settings of model parameters and hyper-parameters. In general, model parameters are those parameters which are directly connected with model, such as optimizer, learning rate, momentum and loss function. While hyper-parameters are those which are used to tune model in order to achieve optimal result for specific model architecture and model parameters, such as number of neurons, activation of layer, amount of dropout, regularization type and penalty(if used), validation split and batch size. One important point for training is that, to make training more general, using of shuffle makes network more general. In other word, internal shuffling of training and validation data-set during training presents more general scenario to model, which helps model to learn general concept for task-achievement. Now, of course, for different settings model have different set of parameters, which makes them to learn information in different amount, but that specific specific setting only can be achieved by analyzing the contribution of model parameter or hyper-parameter in success of model. During model training, mean absolute error also calculated as performance metric.

4.0.1 *Fully connected network(FCNN)*

First of all, the model architecture is selected and and then network is trained over unnormalized training and target data, for mean squared error loss function(MSE), for three different mini-batch sizes(128, 256, 512), for fifty epochs(iterations) and attempted to figure-out best optimizer.

From Fig. 4.1a, one can compare model performance for two optimizers with default settings and for 3 different mini-batch size. For this particular model it is clear that smaller mini-batch size NN model converges faster, while for the case of larger-ones it is unclear



(a) Loss comparison

(b) Overfitting in comparison(Mini-batch:128)

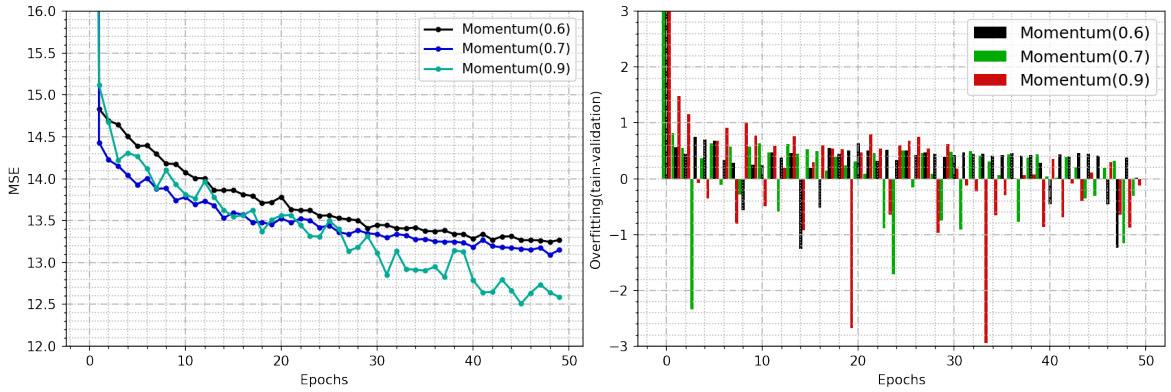
(c) Overfitting in comparison(Mini-batch:128)

(d) Overfitting in comparison(Mini-batch:512)

Figure 4.1: Model performance comparison for two different model parameter selection: Adam and Nadam optimizer for MSE loss function for three different mini-batch batch size

whether it converges like smaller-ones or not. However, only convergence does not determine the goodness of model. Closeness of validation loss along with maximal convergence defines best model. Fig. 4.1b, Fig. 4.1c and Fig. 4.1d present comparison for amount of overfitting for two different optimizers for three batch-size, from which it is clear that, larger mini-batch size increases generalization, and Adam optimizer provides better generalization. As discussed before in § 2.3.4, in some case SGD optimizer performs better than Adam optimizer. Therefore, Fig. 4.2 depicts trial for use of SGD optimizer for this specific task, from which it is clear that it makes model lossy and with increase in momentum clearly it increases model performance.

Fig. 4.3, indicates the evaluation of model using second type of evaluation data, which are created by considering average of distance between successive photons. For example, in Fig. 4.3a, all six sub plots are for special case of average of distance between successive photons. The average would not be every time integer. Therefore, if the digit after decimal is greater than 5 then sample is categorized into next integer count. For example, if one randomly creates 1600 samples, by adding 0 to 60 or 160 photons per sample, then one might have very photon-clumpy samples, as number of photons added are increasing in a sample. After creating all of these samples, there are also samples in which average of successive distance between photons are, for example, below 0.5. All these samples are cat-



(a) Loss comparison

(b) Avg overfitting over training

Figure 4.2: Model performance comparison using SGD optimizer with MSE loss function for different settings of momentum and learning rate values with constant mini-batch size(512)

egorized as Avg<0.5, and above 0.5 to 1.5 they are in similar way categorized(0.5<Avg.<1.5), as shown in Fig. 4.3a with uncertainty considerations, as title of each sub-figures.

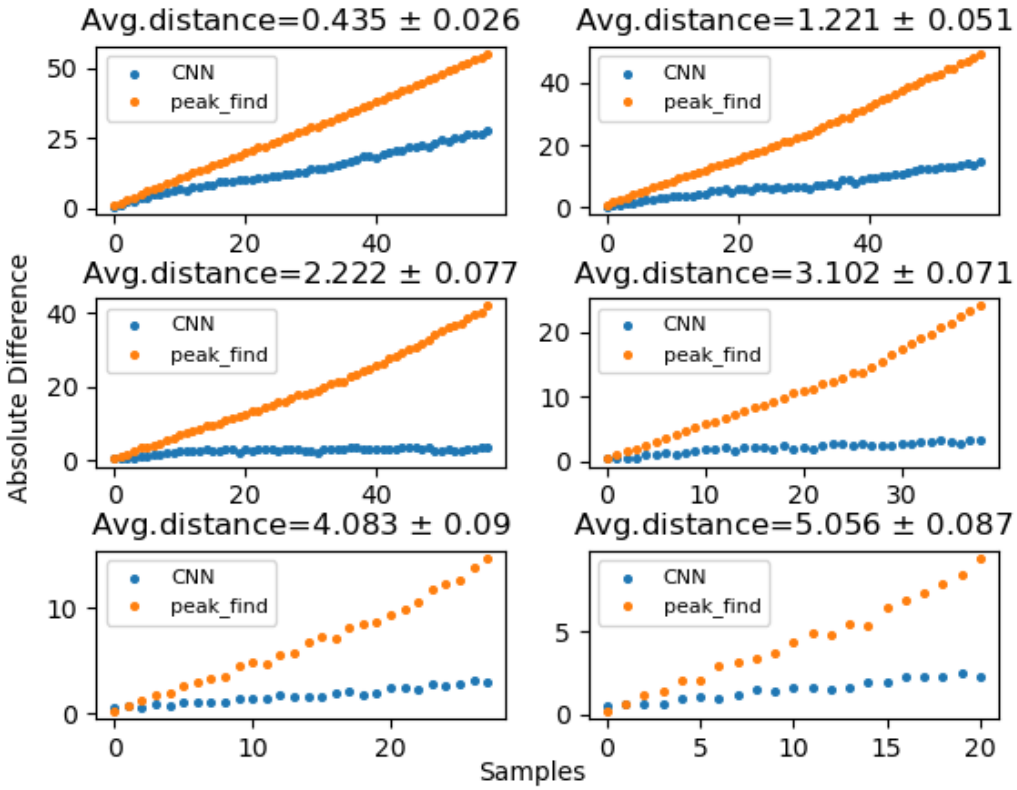
From Fig. 4.3b, it is clear that for average distance between photon from successive photon, from smaller to larger magnitude, difference between model prediction and ground truth is decreasing. In other words, when photons are well scattered then NN works well, while in scenarios like complete overlap of photons, NN has higher uncertainties with higher prediction error, but in both cases NN predictions are closer to ground truth than of peak find. This indicates that NN works well for photon rate calculation.

In this work effect of normalization is also studied in the initial trials. Fig. 4.5 presents results for performance comparison when Adam and Nadam optimizers are used with normalized data for FCNN model training. Normalization is used because in general it makes training smooth, since passed gradient would not be large, which means if model is trained for shorter shorter period of time with least fluctuated training then repetitive use of model would have least chance to give different results or predictions. Fig. 4.5 Fig. 4.5a, Fig. 4.14b and Fig. 4.5c presents least fluctuated training, and reduces loss value and amount of total overfitting over entire training, in which Adam optimizer with learning rate 10^{-3} and mini-batch size 128 has better performance, stability and least overfitting. However this is not always the case, because ultimately training depends on model architecture, parameters and hyper-parameters. And for simplest case here unnormalized data is used.

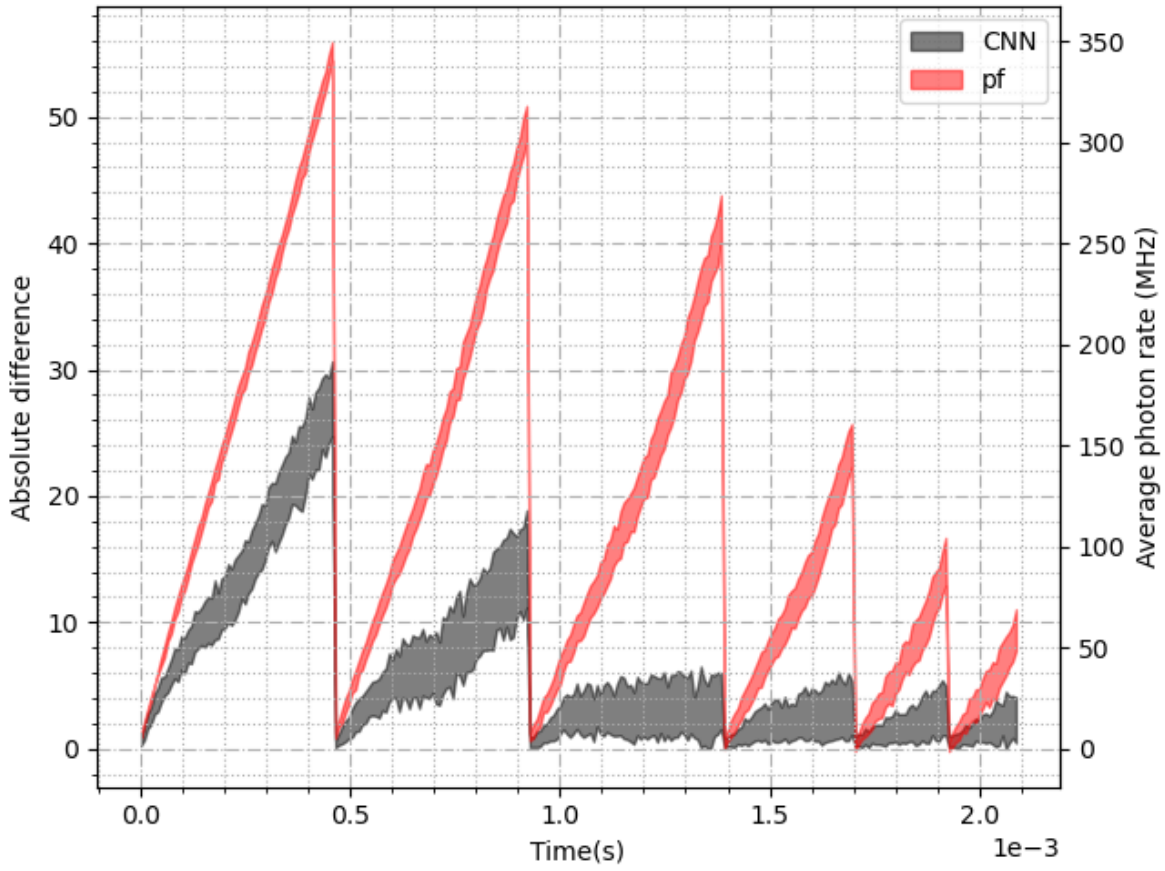
Then convolutional Neural network is tried in order to check effect of CNN on results, and for that following model is used for training evaluation and testing.

4.1 CNN, LSTM AND GRU

From, FCNN, it is clear that Adam optimizer with learning rate 10^{-4} with mini-batch size 128 provides best results so far. CNNs are generally used for vision task and some of the sequential task. Therefore, investigation of CNN model training in this case is important.

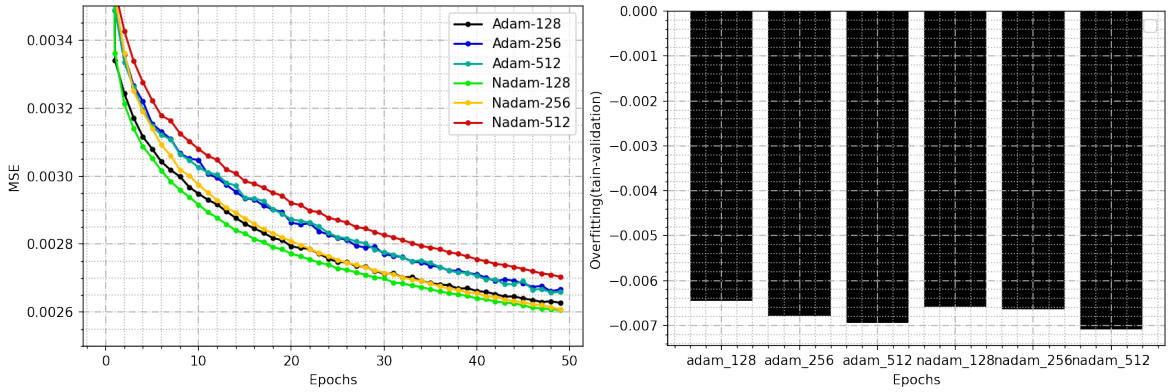


(a) Evaluation of model which is trained without normalization, by using Adam optimizer and batch size(128)



(b) Evaluation of model which is trained without normalization, by using Adam optimizer and batch size(128)

Figure 4.3: FCN evaluation



(a) Loss comparison

(b) Avg overfitting over training

Figure 4.4: Model performance comparison between two optimizers with three different mini-batch size when data are normalized.

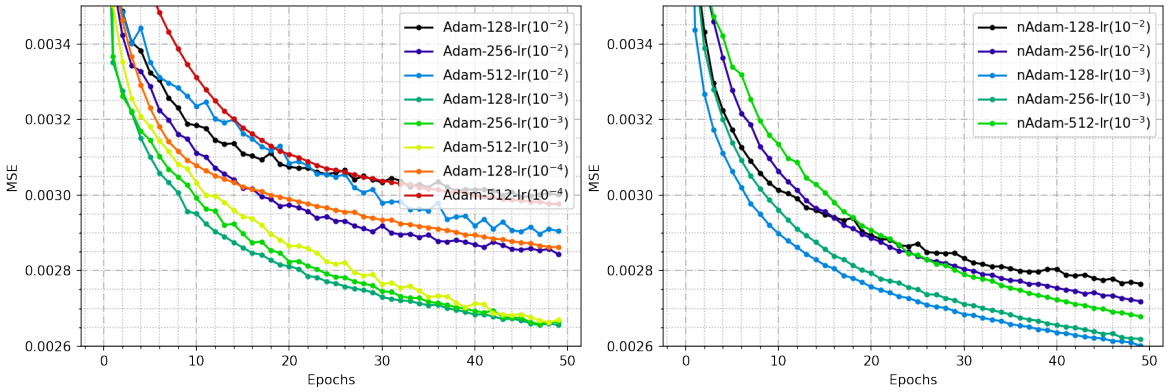
4.1.1 *CNN evaluation*

In CNN architectures, we preferred simplest models with kernel size 3. Use of CNN has improved predictability of model. For CNN, absolute error has decreased by factor of 10 for samples with average distance between peaks< 0.5, 5 for average distance between peaks 2 and for the rest of the cases uncertainty has been improved. Also, for all 5 cases of average distance between peaks, CNN model has almost stable uncertainty in prediction in entire range of rate, but for second case(average distance between peaks 1), it is better for peak extraction at lower rate comparing to higher rate. (see Fig. 4.8)

4.1.2 *LSTM-GRU evaluation*

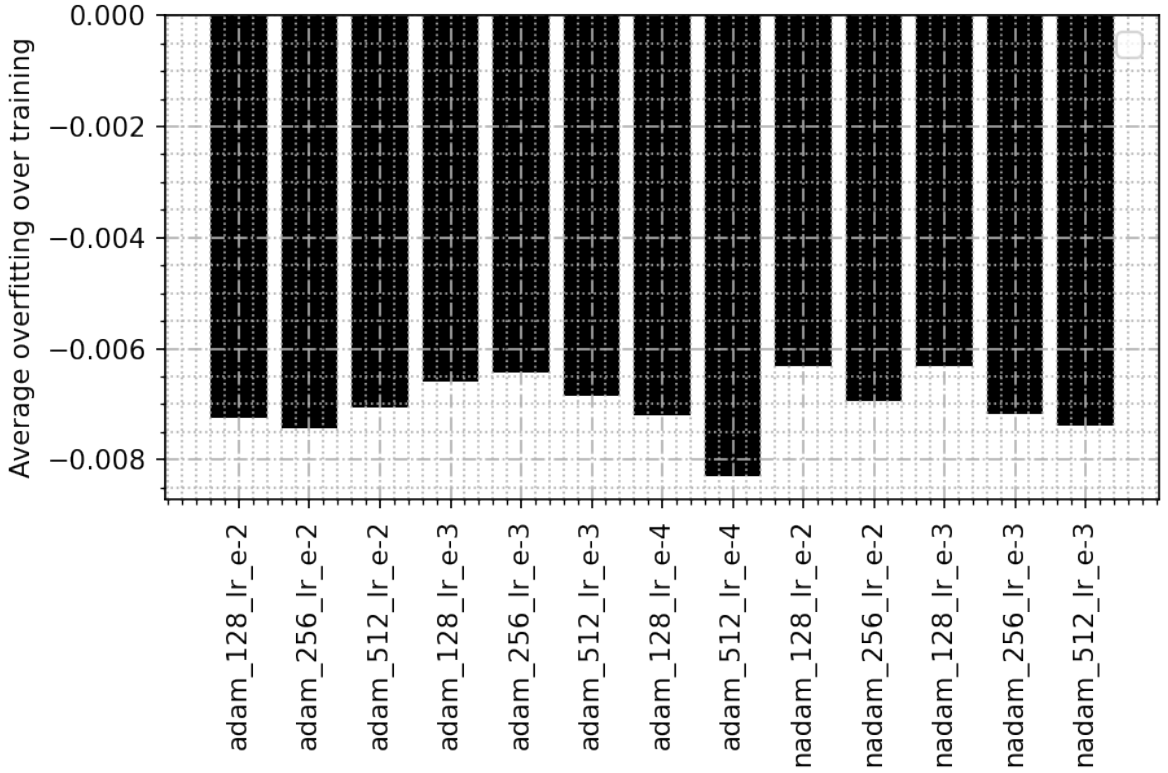
Comparing model evaluation of LSTM-GRU models, GRU has better ability for peak extraction than LSTM when average distance between peaks is less than 0.5, by factor of 8, while for average distance between peaks 1. Considering all five cases of average distance between peaks, it is clear that for average distance<0.5, LSTM predicts almost equally at each rate, while GRU has larger uncertainties at lower rate while smaller at higher rate. For average peak distance 1, GRU predicts with larger uncertainties over the entire range of rate while LSTM has higher uncertainties at lower rate and at higher rate it has smaller uncertainties. Overall, with increasing spacing between peaks, LSTM-GRU model average prediction improves but on the other hand model uncertainties are increased(see Fig. 4.9 and Fig. 4.10).

Fig. 4.11 presents, comparison of model evaluation over evaluation dataset, from which it is clear that charge integration predictions(yellow errorbar) are diverging from ground truth with increment of rate by overprediction of rate, while Neural Network models such as CNN, LSTM, GRU are predicting closer to ground truth than Charge Integration even at higher rate, but they are underpredicting in between 300-400 MHz. Comparing LSTM and GRU performance, overall LSTM is better than GRU, but they both have identical



(a) Loss comparison

(b) Loss comparison



(c) Avg overfitting over training

Figure 4.5: Model performance comparison between two optimizers with different settings of learning rate and three different mini-batch size when data are normalized.

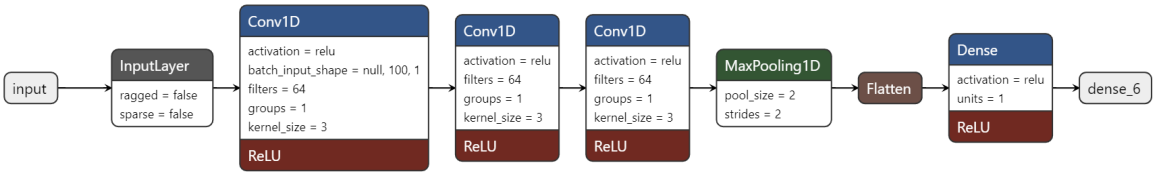


Figure 4.6: CNN model

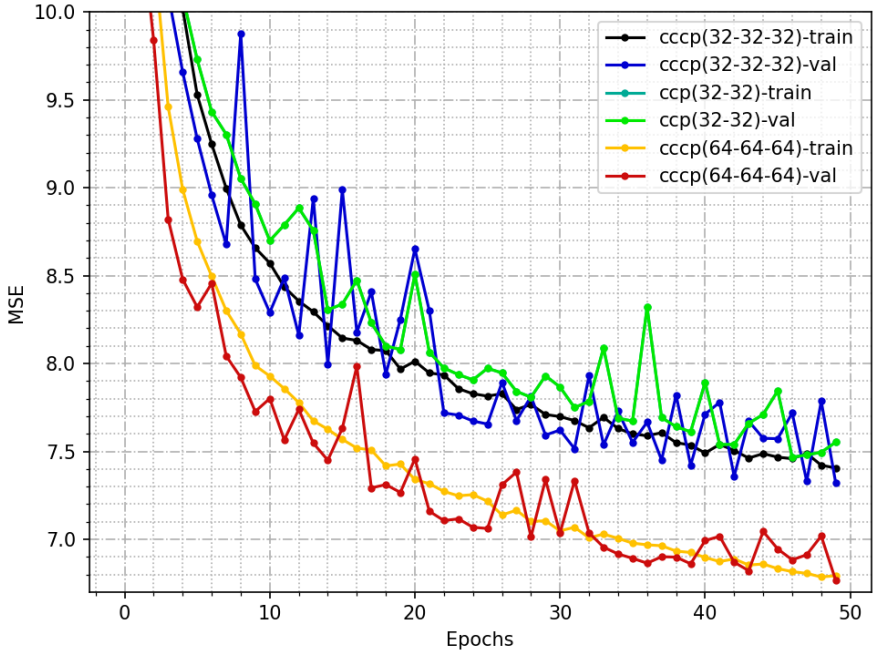


Figure 4.7: Model training of selected CNN architectures

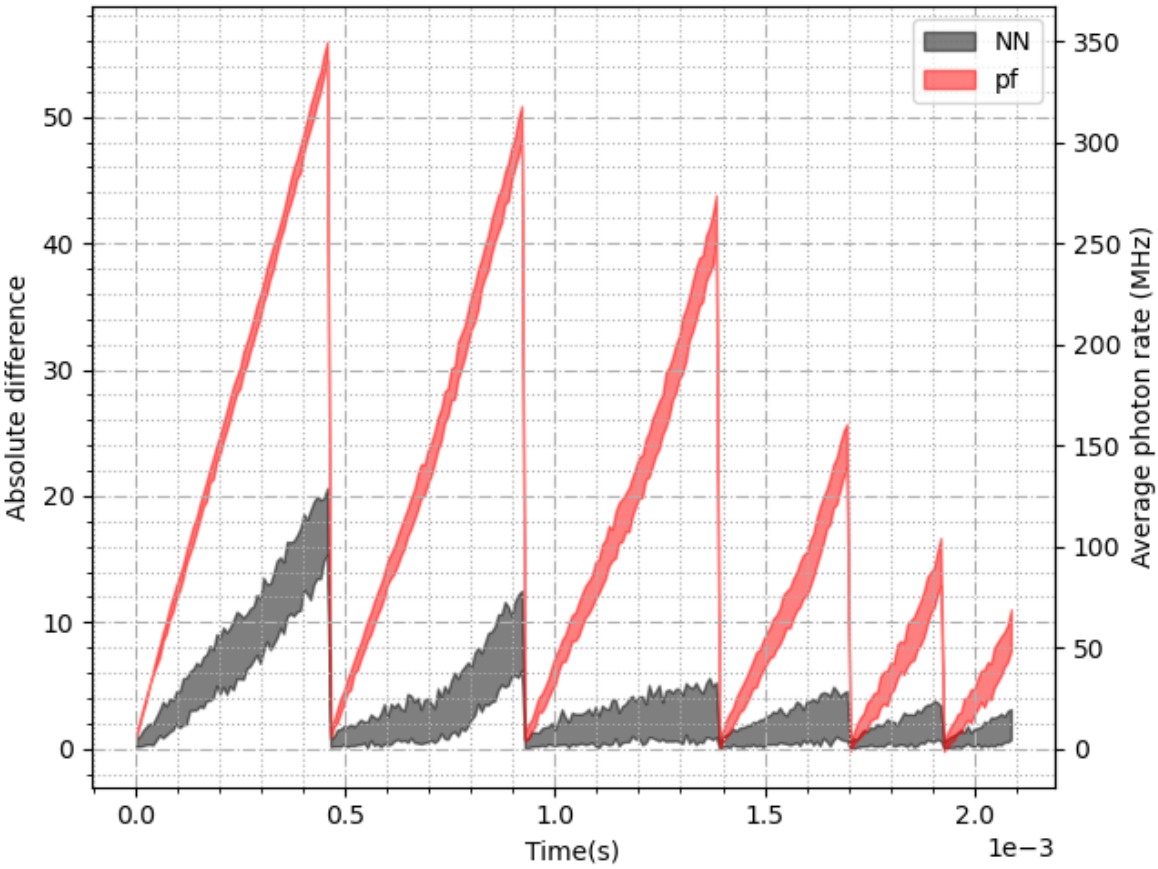
predictability in 300-400 MHz range. Among all three types of NNs, since, CNN is closest to ground truth, CNN architectures are used more, but LSTM-GRU are also used often.

All these models were also tested on data with higher transmittance of photons, and predictions are compared with other two methods, Charge integration and extrapolation of first six points from peak find.

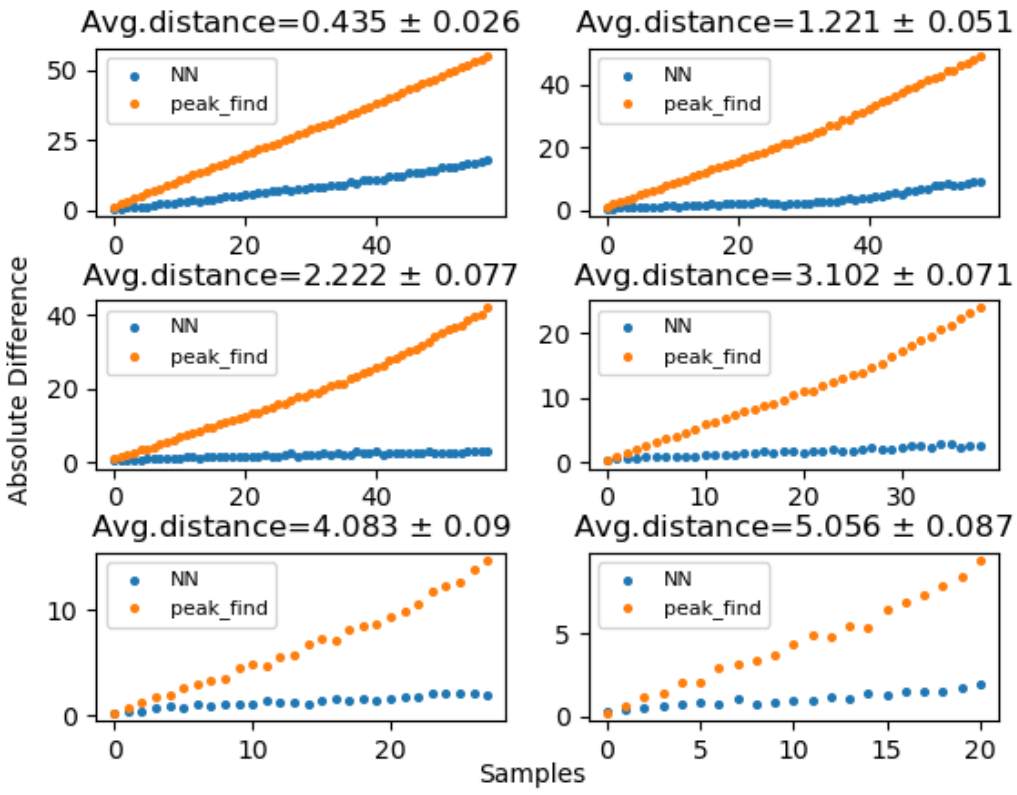
For amount of learning with constant data size, it is also important to check dependency of better model prediction over sample size, and for that here we tried sample size of 500 time-stamps. Six different configuration of model are trained without normalization of data for sample size 500, and evaluated over set of samples with six different configurations of average distance between peaks. From evaluation result comparison of CNN models, there is no large impact of model complexity on prediction. Evaluation results are almost identical, but increasing complexity has increased time of training. The peaks in case 3 and 4 are due to samples with different rate with identical average distance between photon peaks. For LSTM-GRU, evaluation results conclusion is completely different than for sample size 100. Their overall behaviour for all cases are similar, but with slight higher uncertainty for case 1 and 2 for GRU than LSTM(see Fig. 4.13).

4.1.3 *Model Testing*

After evaluation of all different types of models, prediction test is important to determine true predictability of model. Here, we used test data generated form laboratory experiment for different transmittance, in increasing order which gives higher rate photon samples. The only thing in control is the amount of transmitted photons and size of the sample, which are 100 and 500. Without normalization of datasets, trained models are tested over these dataset, and prediction of models are compared with each other, Charge Integration

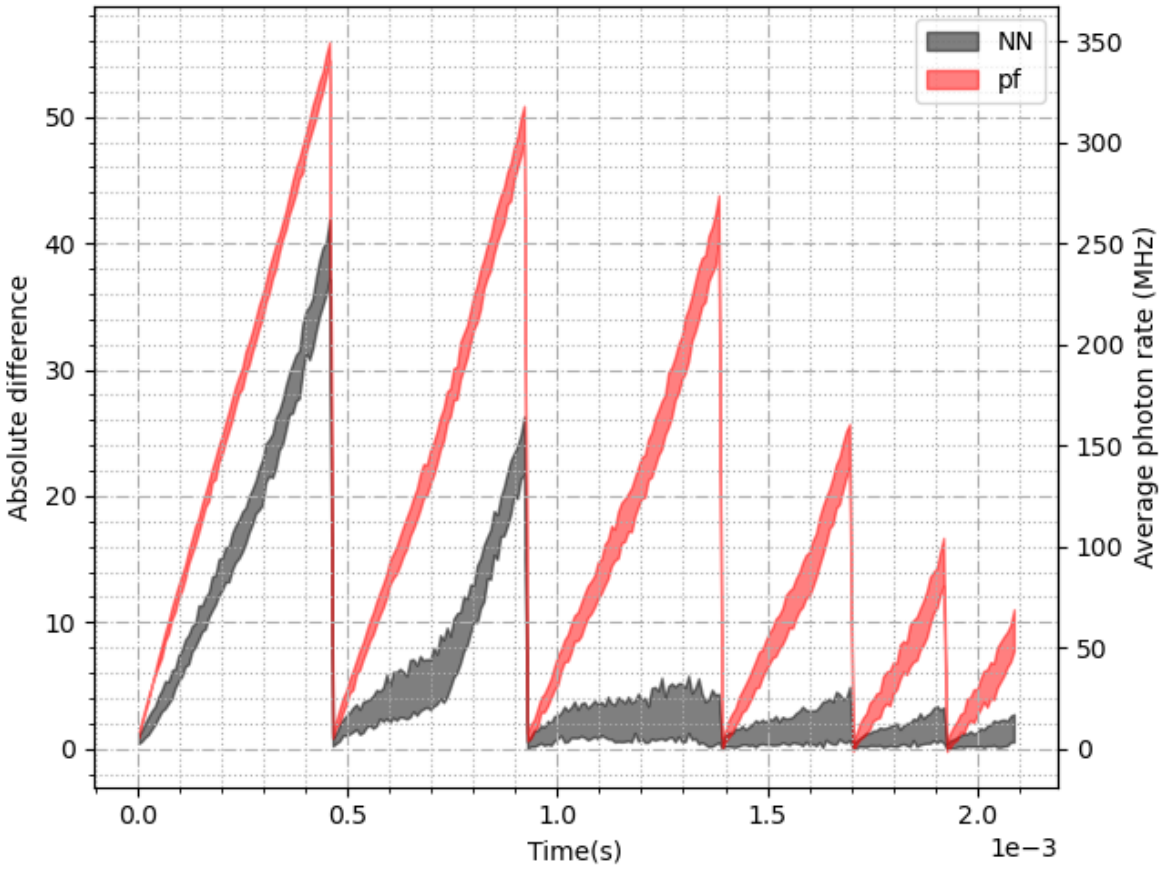


(a) CNN model evaluation plot

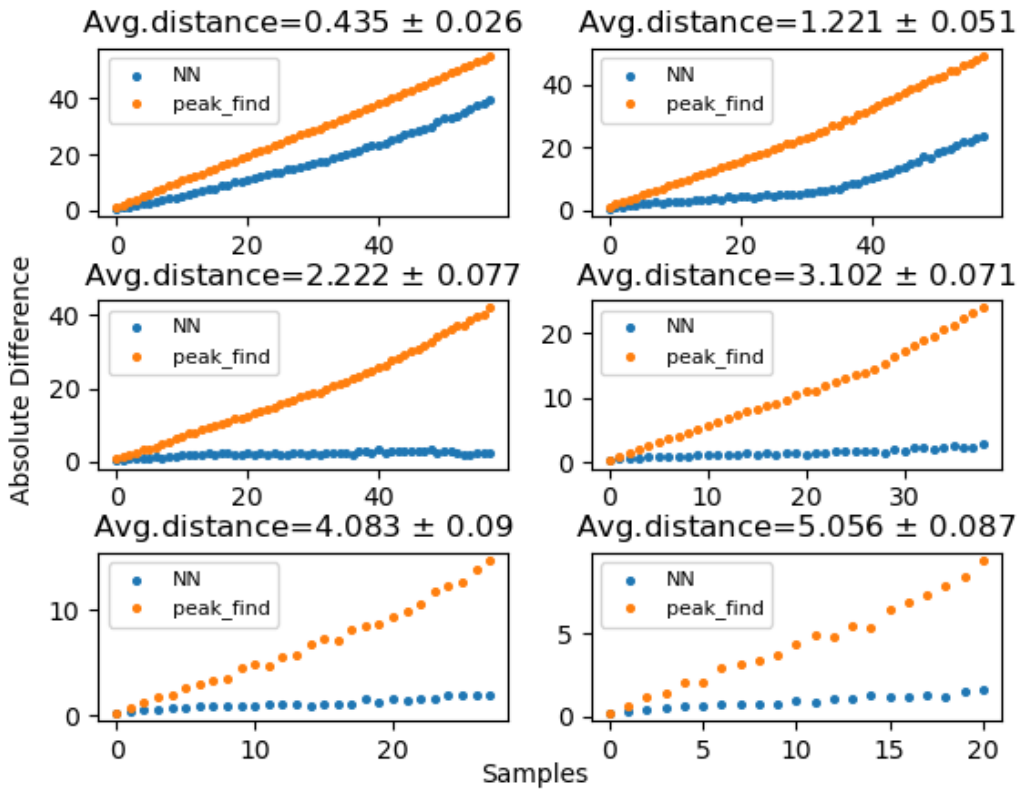


(b) CNN model evaluation plot

Figure 4.8: CNN model evaluation

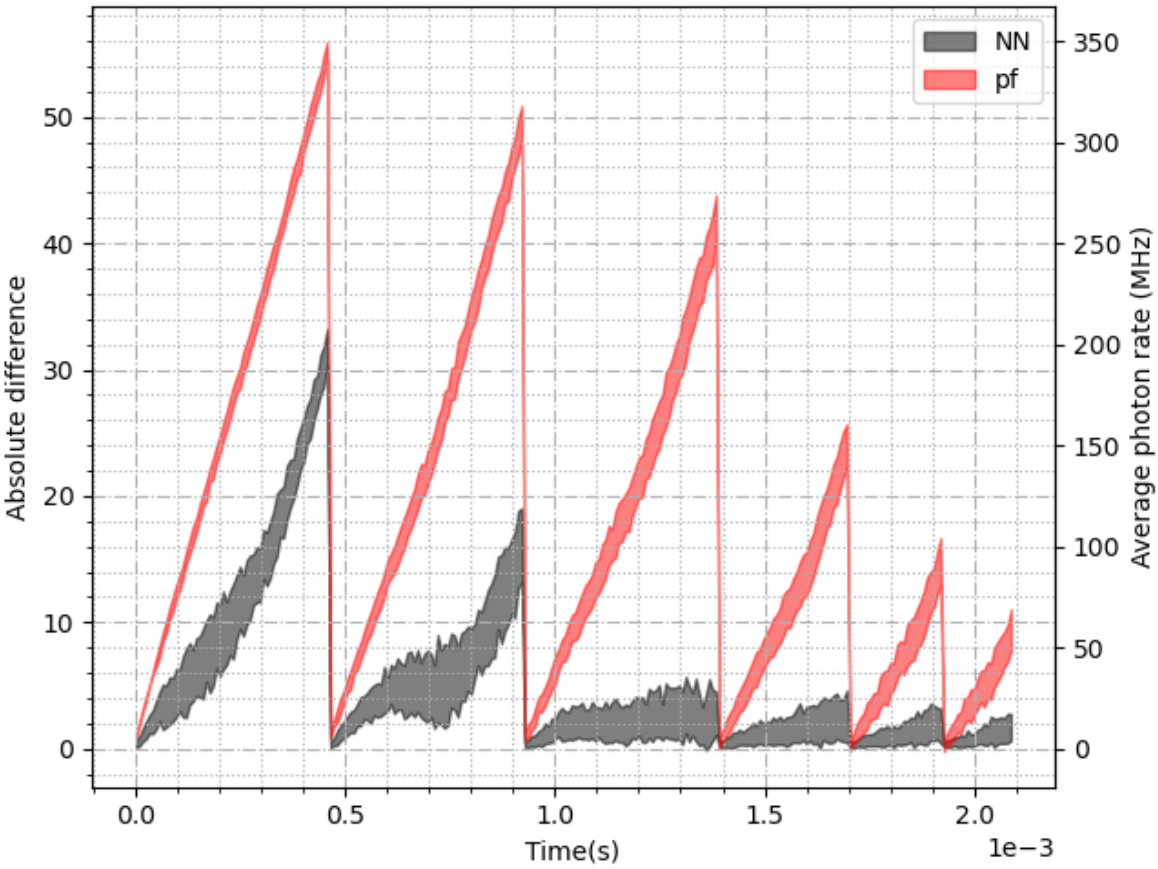


(a) Bidir-LSTM model evaluation plot

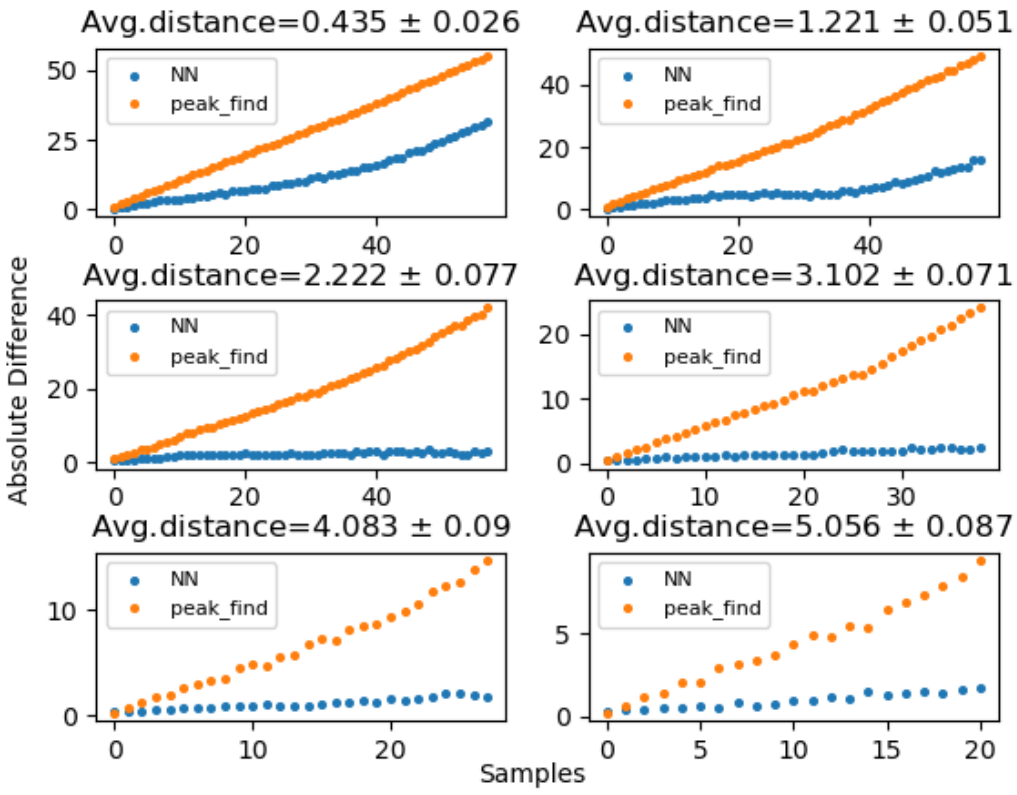


(b) Bidir-LSTM model evaluation plot

Figure 4.9: LSTM evaluation



(a) Bidir-GRU model evaluation plot



(b) Bidir-GRU model evaluation plot

Figure 4.10: GRU evaluation

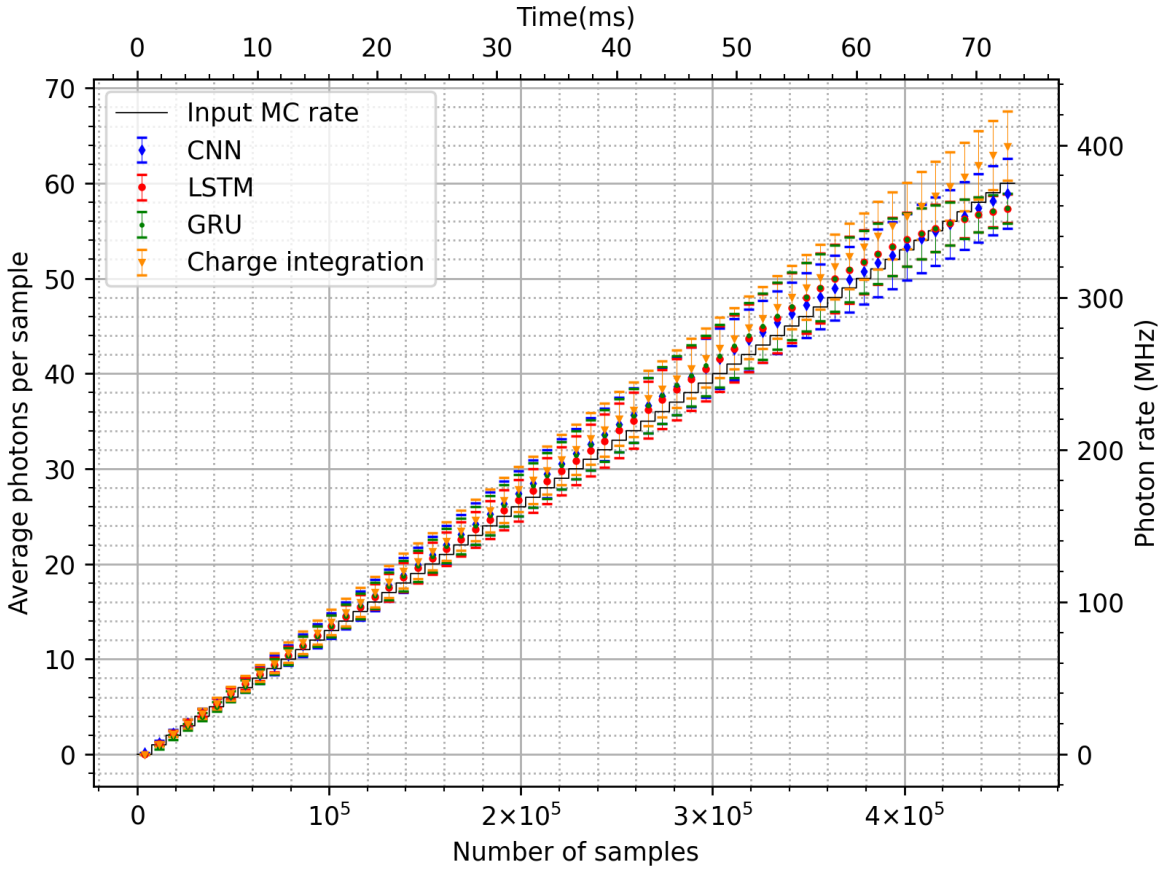


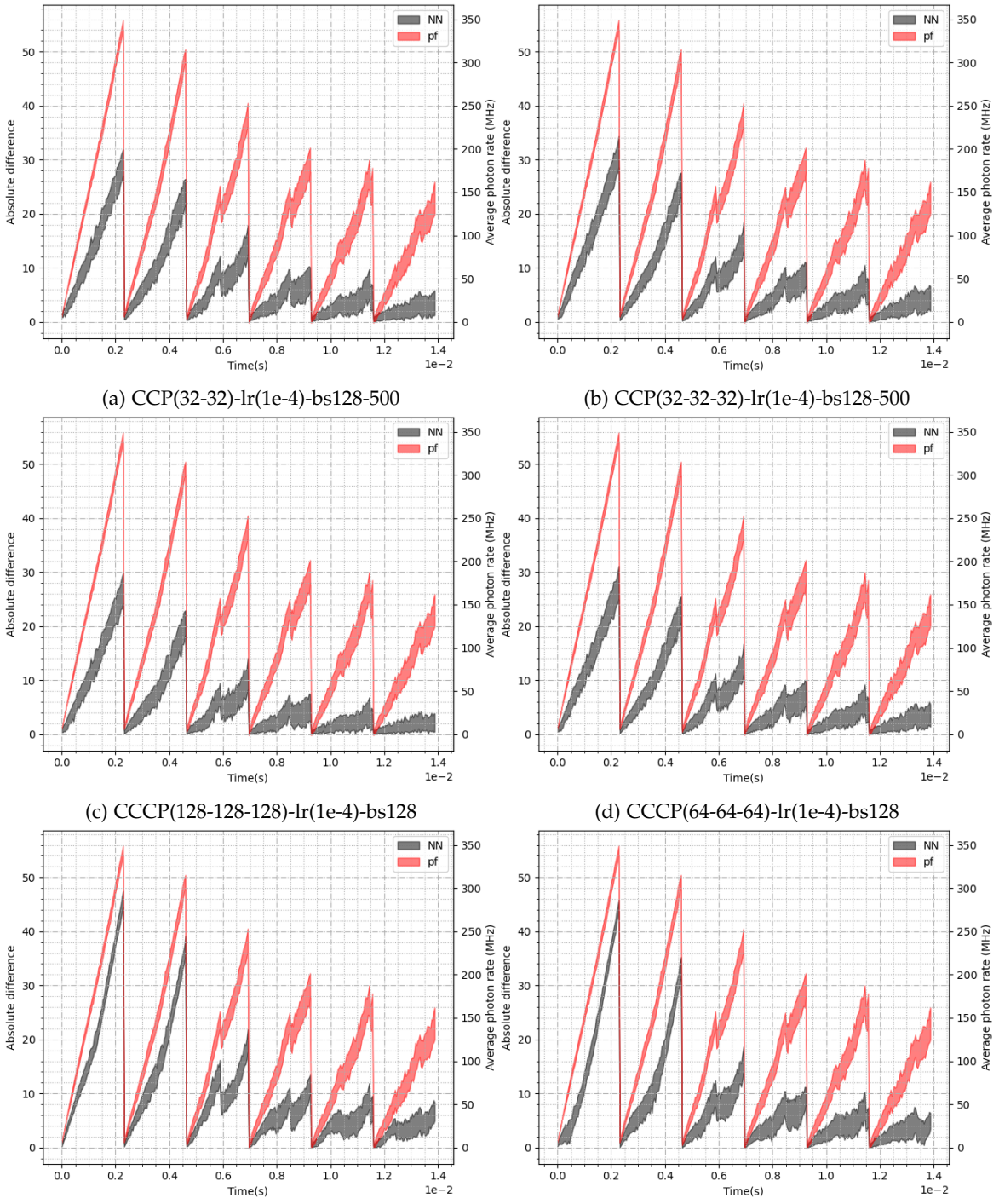
Figure 4.11: Prediction comparison

and extrapolation of first six data points from peak find. In Fig. 4.13, first column of image is for sample size 100 and second one is for sample size 500. All models and methods are are sensitive to sample size(for sample size 100 model and method predictions have larger uncertainties compared to sample size 500 at each transmittance). Although NN models are least uncertain in both cases compared to Charge Integration. Peak finding method determines rate with best results at lowest transmittance while it suffers from under-prediction at higher transmittance. Fig. 4.13a presents different models, their Floating point operations(undergoing calculations), and their connection with sample size.

Similar configuration was used for photon rate measurements form Sirius using IceAct telescope and all the methods have been compared(see Fig. 4.14). From mathematical calculations provided by Dmitry Malyshev for photon flux from Sirius, expected photon rate from Sirius is 15MHz, when 0.1nm filter is used for measurements. All the methods are predicting rate from Sirius quite close to each other, with some off-set for background measurement. Fig. 4.14a is relative error comparison for all methods and Fig. 4.14b shows relative error comparison of all methods when Sirius in field of view, from which it is clear that NN models have least relative error uncertainty while Charge Integration has the higher uncertainty in relative error.

From all these attempts, following points can be concluded,

- Higher learning rate results fluctuated training, i.e, unstable model



(a) CCP(32-32)-lr(1e-4)-bs128-500

(b) CCP(32-32-32)-lr(1e-4)-bs128-500

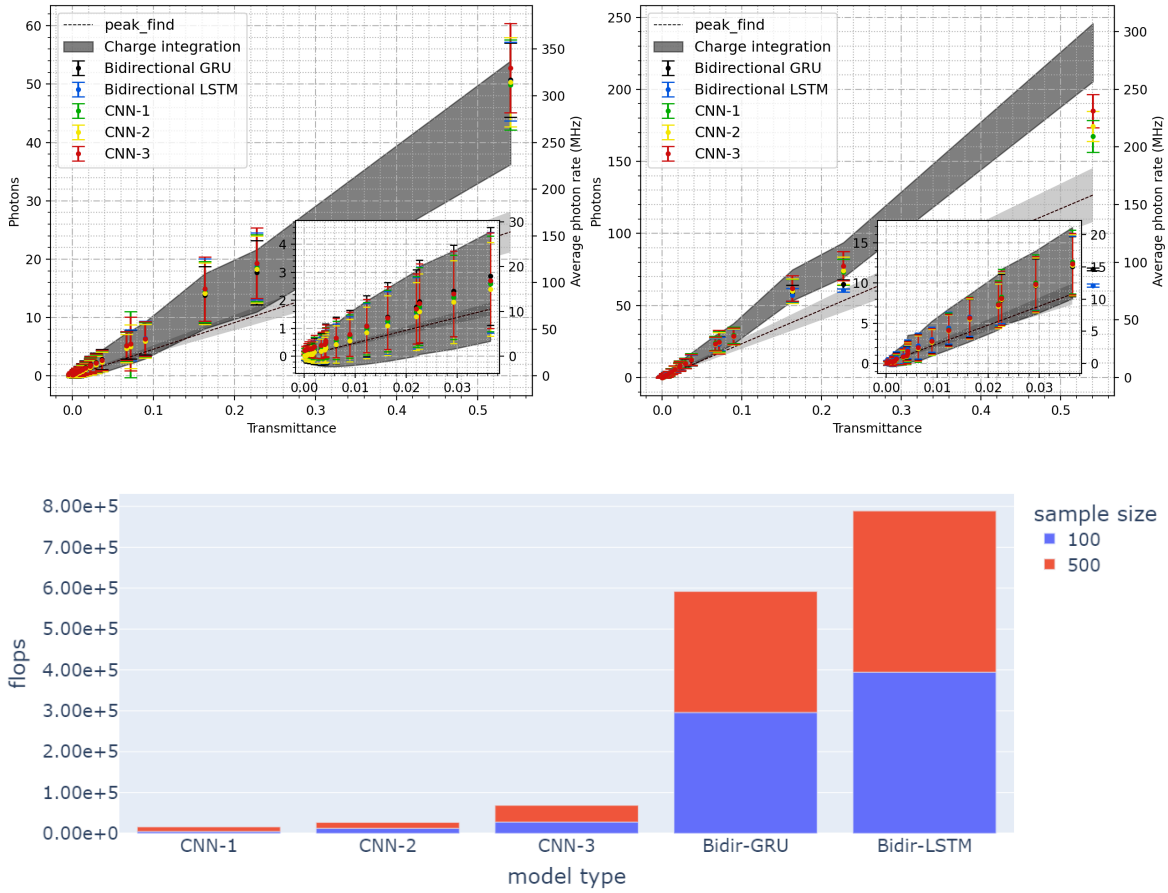
(c) CCCP(128-128-128)-lr(1e-4)-bs128

(d) CCCP(64-64-64)-lr(1e-4)-bs128

(e) Bidir-LSTM(128)-lr(1e-3)-bs128

(f) Bidir-GRU(128)-lr(1e-3)-bs128

Figure 4.12: Model evaluation of CNN(with different configurations) and LSTM and GRU when sample size 500 is used



(a) Model complexity comparison

Figure 4.13: Model prediction comparison over data with higher transmittance

- Adam optimizer is the best suitable choice for our dataset.
- Larger mini-batch size reduces fluctuations during training ,but also can make model undertrain for smaller number of epochs for training
- Learning from samples depends on combination of learning rate and mini-batch size. In so far in our case small batch with small learning rate improves results.
- Convolutional neural networks, LSTM and GRU models are working best for our case so far.

So far we have used datasets, which have samples with number of photons from 0 to 60 i.e, 0 to 375MHz. We concluded that Neural Networks are sensitive to used sample size. Therefore, it is required to check behavior of Neural Networks for higher rate scenario. Therefore, we created another set of MC data which contains examples with number of photons per sample from 0 to 160 i.e, 0 to 1GHz, and first tried to compare model with 2 convolutional layer and evaluated model, then tested model over lab measurements.

We have used first model configuration with two convolutional layers, with different filter size, in ascendind and in descending order, and also used Resnet architecture with increasing depth of network. After training these models with different configurations of

batch size and learning rate we first compare the performance of models and then evaluated and at the end we compared predictions of each model over laboratory measurements and also compared each prediction with charge integration method and peak find extrapolation of first six prediciton points.

Fig. 4.15a and Fig. 4.15b indicate the loss curve of different ResNet configurations, in which set of train and validation curves are for each configurations are from models of three different depths. 'Block1' is the shallowest and 'Block3' is the deepest among three. All three models were trained using 3 different mini-batch sizes 128,256 and 512. While Fig. 4.15c, is loss curve comparison of two different models with two convolutional layers, first one is with first layer thickness of 4 and second layer thickness of 2, while second one is with first layer thickness of 4 but second layer thickness of 32. All the models are trained using 3 different mini-batch sizes 128,256 and 512. By analyzing Fig. 4.15, it is clear that with increasing number of layer model loss decreases, i.e, ResNet models has low loss, even with the use of large Mini-batch size(512). Also, out of all three mini-batch sizes the lowest loss can be achieved while using mini-batch size 128, while it is higher in majority of cases for mini-batch size 512.

In Fig. 4.16, the used model architectures are presented. Here, we create ResNet architectures in such a way that we create mainly three stages, in which each stage consists of bunch of convolutional and identity branch. In First stage we used thickness of branch in 2,4,8 order. Then we added few identity branches with same thickness configuration(in our case it is 3 for shallowest case), then we add next stage with doubled thickened of layers, and we also double the thickness for third and last stage addition. Overall, we use more number of identity branches in first stage while we reduce number of identity branch, as we further add stage. But the beginning of each stage is convolutional branch, which is only responsible for size reduction of sample in Resnet. In this way we create models with 3 different deepness, with block-1 which adds only one set of identity blocks, and for stage-1 there are four identity layers in one block, for two blocks it simply doubles. For second stage it has block with two identity branches. For stage 3, it's block has only one identity branch. In this way we have created three different ResNet architectures with increasing depth, and used them for further training, evaluation and prediction.

We trained these models for different combinations of learning rate and mini-batch size. Then, we tried to evaluate all of these models using evaluation data in order to figure out model behaviour for different scenarios of photon clumping such as average distance between successive photons as 0,1,2,3,4 and 5.

In evaluation stage of all three models with different configurations, we have found that for less complex model architectures(Fig. 4.17a and Fig. 4.17a), use of smaller learning rate with larger batch size makes model more accurate, and for less complex models quite opposite case is true in many cases, i.e, complex model with trained by higher learning rate with smaller mini-batch size provide better results. By comparing both cases of CNN with ResNet, it is clear that, CNN have better peaks extraction capability than ResNet with lower average distances between successive peaks. While for higher case they both behave quite similarly. Also, for higher rate the uncertainty in average difference is lower and for

lower rate it is higher. Then we tried to test model over lab measurements with different transmittance of photons.

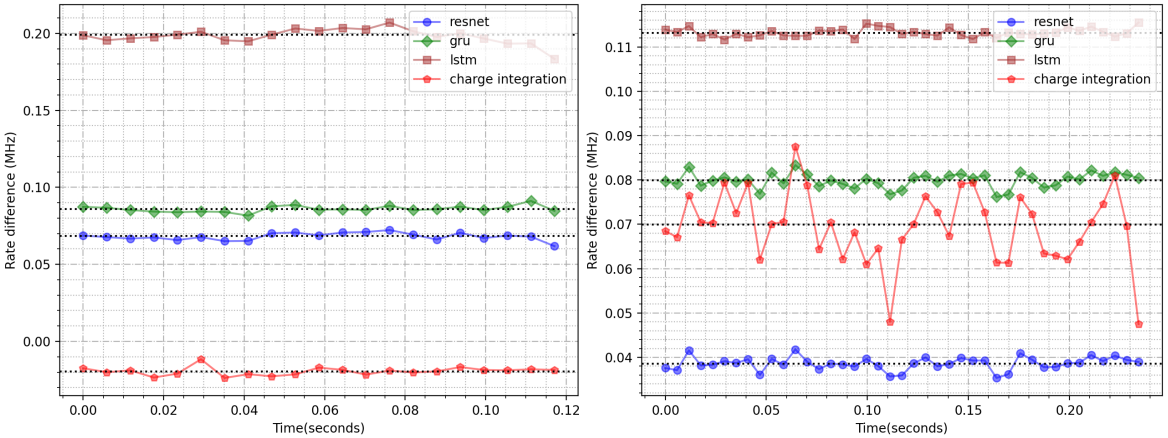
From Fig. 4.19a and Fig. 4.18, one can determine which model is better. For lab measurement, since we do not have access to ground truth, it is difficult to determine directly, which model has prediction closer to ground truth. Therefore, here three models are selected in such a way that one model is predicting almost identically to Charge Integration, and rest are not. Now, once we have evaluation analysis of model, and prediction, we can approximately determine the ground truth. Considering model evaluation in Fig. 4.18 and order of model errorbars in Fig. 4.19a, approximately, the highest transmittance scenario could be the 1.5<avg<2.5 or 2.5<avg<3.5 scenario in Fig. 4.18, i.e, either average distance between successive photons either 2 or 3 or in between. From this, we can say that ground truth might be laying somewhere around upper edge of Charge integration. While for second last and third last transmittance errorbars, orders are quite close to 3.5<avg<4.5 scenario, and then at the lowest transmittance, the orders are in favour of 4.5<avg<5.5 scenario. Therefore one can say that, for lower to below intermediate transmittance the ground truth could be laying within the Charge integration region, while for the rest of the part it is getting closer to the upper edge of Charge integration. Model-1 is underpredicting at the lowest transmittance, while other two models are predicting close to the peak find method. Fig. 4.19b is the similar comparison but all models are trained by using mini-batch size 512. Again, comparing it with Fig. 4.18, one can say that the highest transmittance scenario could be 1.5<avg<2.5 or 2.5<avg<3.5 case, and then at intermediate transmittance it follows case 3.5<avg<4.5, and at lower transmittance it again follows 4.5<avg<5.5 scenario because ResNet predictions are quite matching with Charge integration by being the closest to peak find extrapolation region among all three models. Prediction comparison among ResNet architectures with different configurations are shown in Fig. 4.22a, Fig. 4.23a and Fig. 4.24a

We also applied LSTM and GRU architectures in same manner as we applied for low rate measurements. We found similar behavior, during testing of model, that model predictions are saturating when transmittance is above 0.1 and rate is above 100 MHz. Therefore we also tried using LSTM and GRU in different manner in which we extract output from each unitcell and then used it for further in neural network. In this way we have obtained predictions very close to charge integration(see Fig. 4.21b). Overall, in prediction LSTM and GRU predictions are very close to each other, and strictly following Charge Integration till 120 MHz, afterwards they are underpredicting compare to Charge Integration, by being close to peak find extrapolation band. We have used this similar architecture for training model over MC generated Sirius dataset, and predicted over time. Comparison shows that both LSTM and GRU are close to each and predicting rate in quite similar way as it was tested on laboratory measurements(see Fig. 4.25), and we also found that at low rate they are predicting with higher uncertainty but at higher rate they predict rate with lower uncertainties compare to Charge Integration.

After that we also trained ResNet models with different settings for training and validation dataset containing rate from 0 to 1GHz, using MC algorithm, on measurements from

Sirius using 30nm filter. We used similar configurations, and tried to match the prediction of models relative to Charge Integration and with predictions over lab measurements.

We notice that with the increase in depth of ResNet architecture, it is clear that model tends to underpredict the samples, while the shallowest model has the predictions slightly overpredicting Charge Integration. At the tail part(low rate scenario), ResNet predictions and Charge Integration predictions are getting closer to each other. Overall, all the models follow the pattern which Charge integration has. In this way we can confirm that models obtained from lab measurements are suitable to use for Sirius measurements.



(a) Night sky background

(b) Sirius

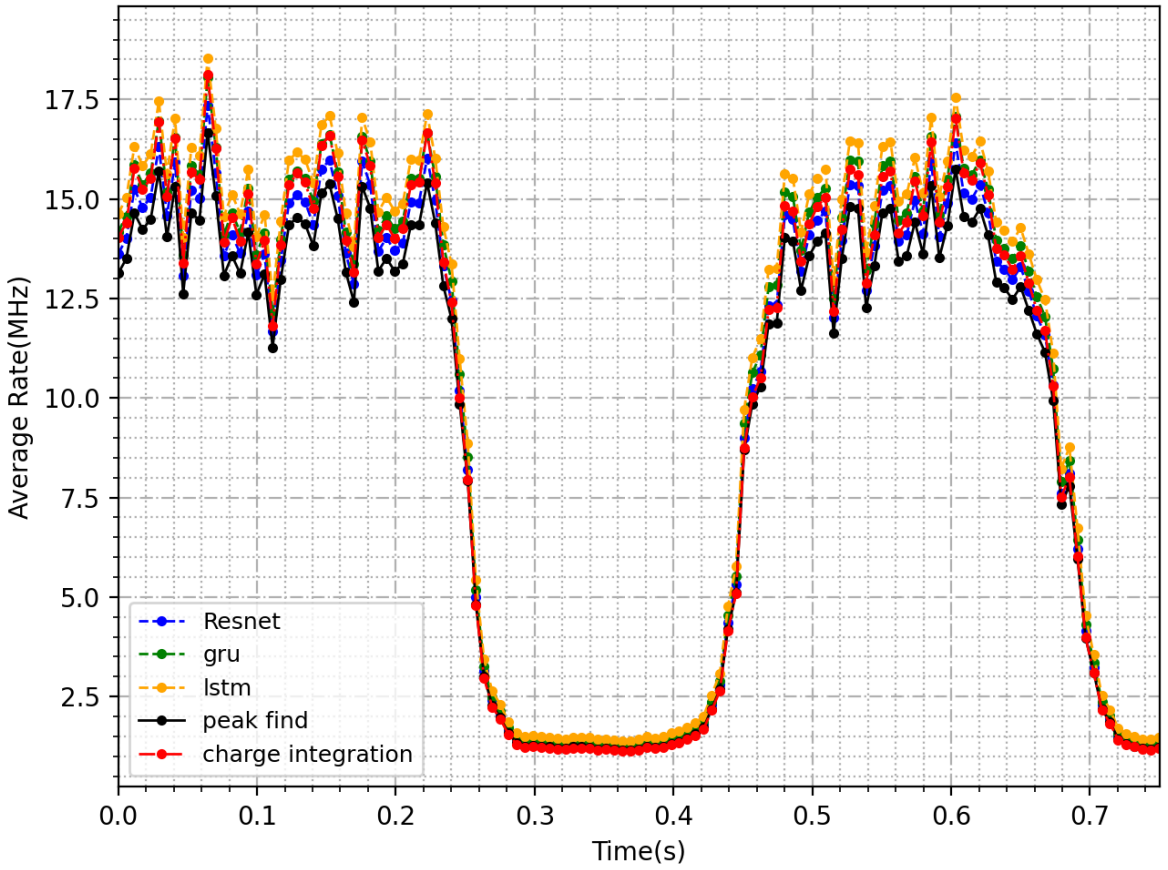
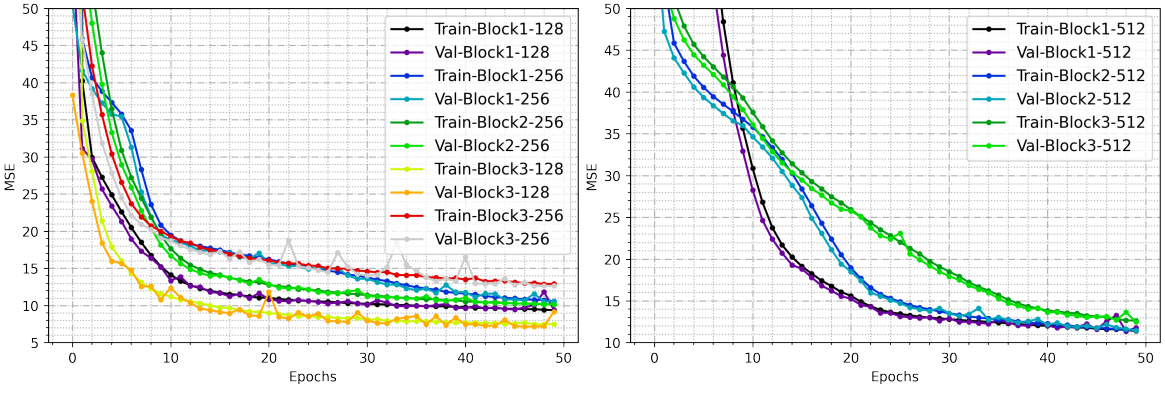
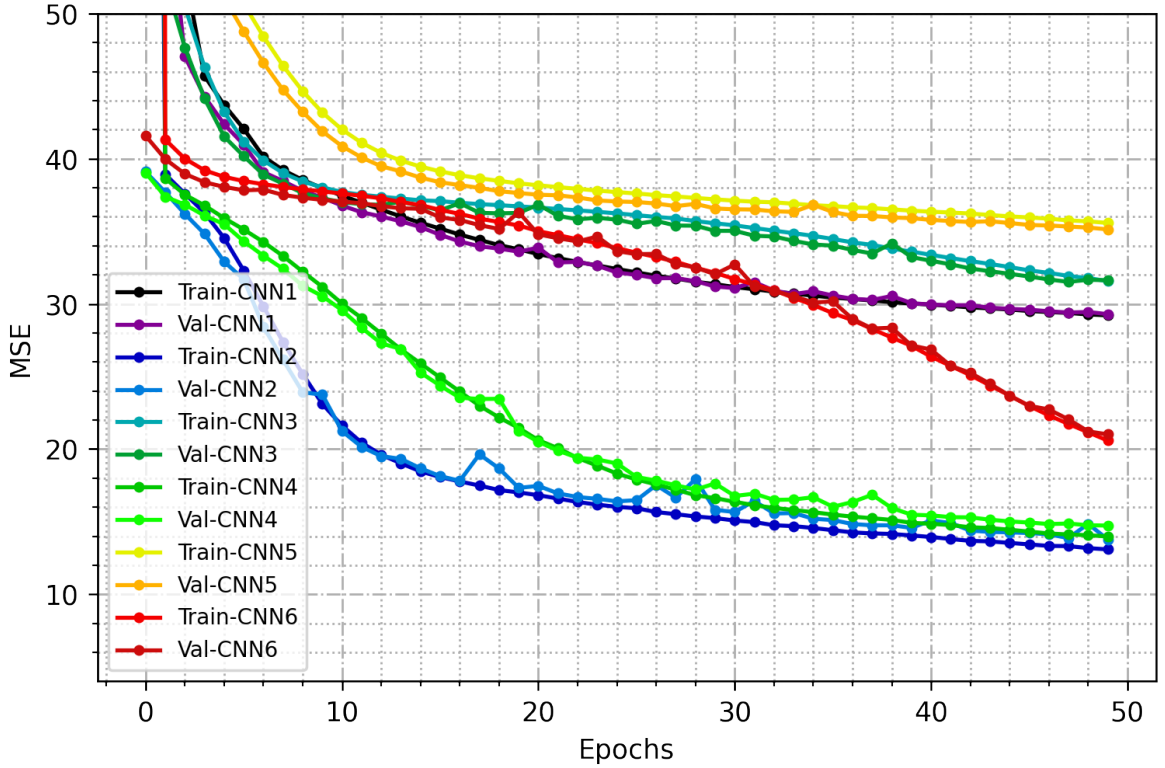


Figure 4.14: Photon rate measurement form Sirius using different methods



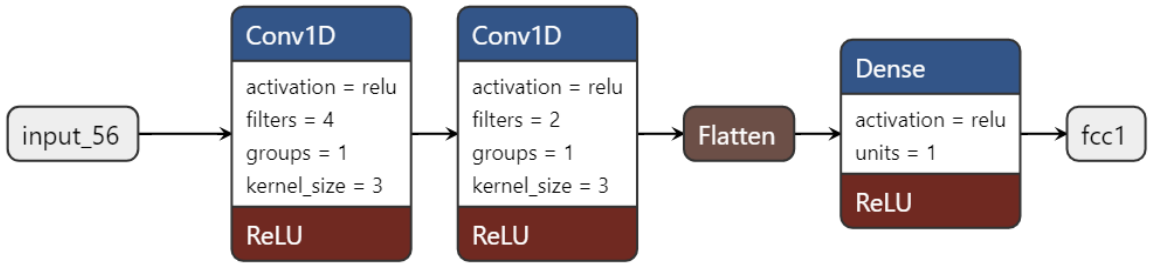
(a) ResNet

(b) ResNet

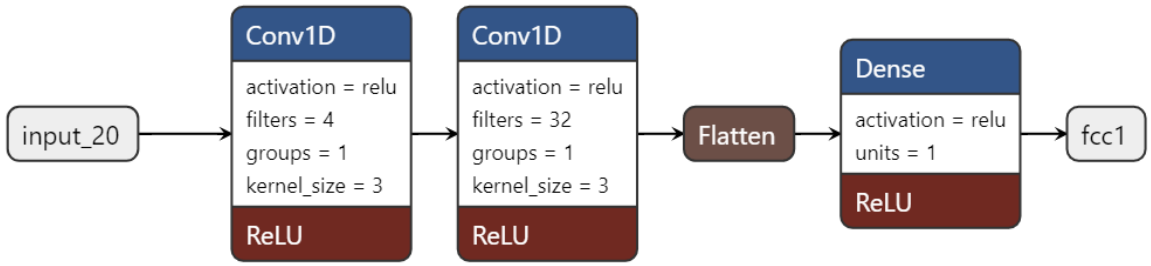


(c) CNN

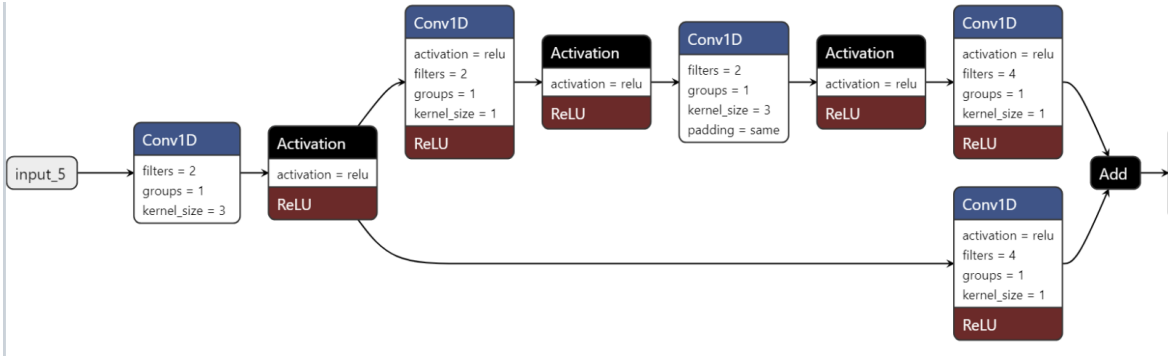
Figure 4.15: Loss curve of different model configurations, CNN and ResNet. For CNN, in Fig. 4.15c, model abbreviation CNN1, CNN3, CNN5 are used when model's first layer is thicker than second one, and others for thickness other way around. For ResNet(Fig. 4.15a and Fig. 4.15b), names, block1 means shallowest, while block3 means deepest out of three. All models are trained using learning rate 10^{-4}, but with different batch sizes.



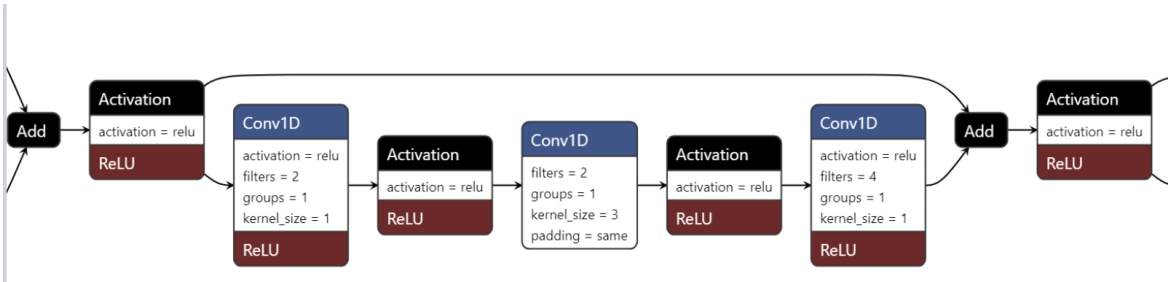
(a) CNN-CC(4-2)



(b) CNN-CC(4-32)

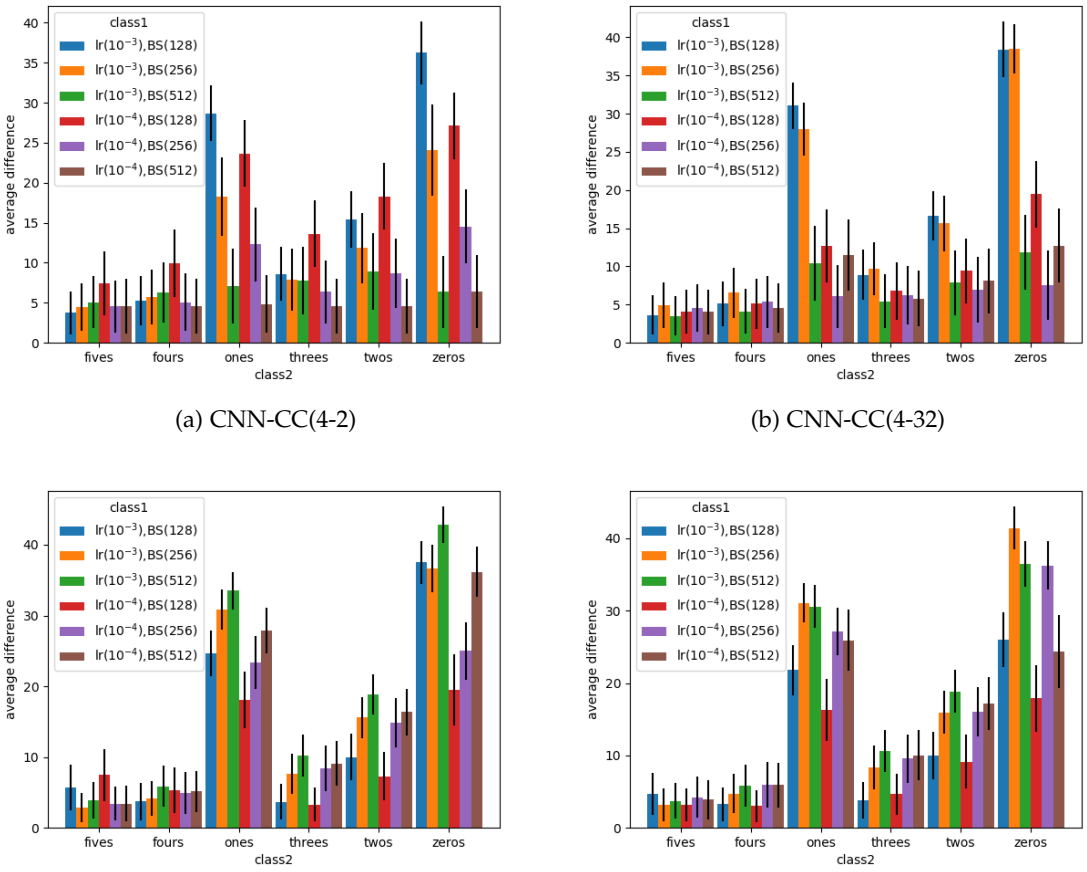


(c) ResNet Convolutional branch



(d) ResNet-identity branch

Figure 4.16: DIfferent model architectures. simplest architectures Fig. 4.16a and Fig. 4.16b, we used every time same. But for the case of ResNet, we have used different combination of convolutional branch(Fig. 4.16c) and identity branch(Fig. 4.16d)



(a) CNN-CC(4-2)

(b) CNN-CC(4-32)

(c) ResNet-block-1

(d) ResNet-block-3

Figure 4.17: Mode evaluations of two types of simple CNNs, one with thinner layer followed by thicker layer(Fig. 4.17a) and other with thicker layer followed by thinner(Fig. 4.17b), and of two types of ResNet models, Fig. 4.17c(shallow) and Fig. 4.17d(deeper) with different BS(mini-batch size). class1 in each plot contains different training configurations, and class2 contains average distance between successive photons in MC evaluation data. Y-axis in each plot is for average difference between prediction and ground truth, over entire range of rates used.

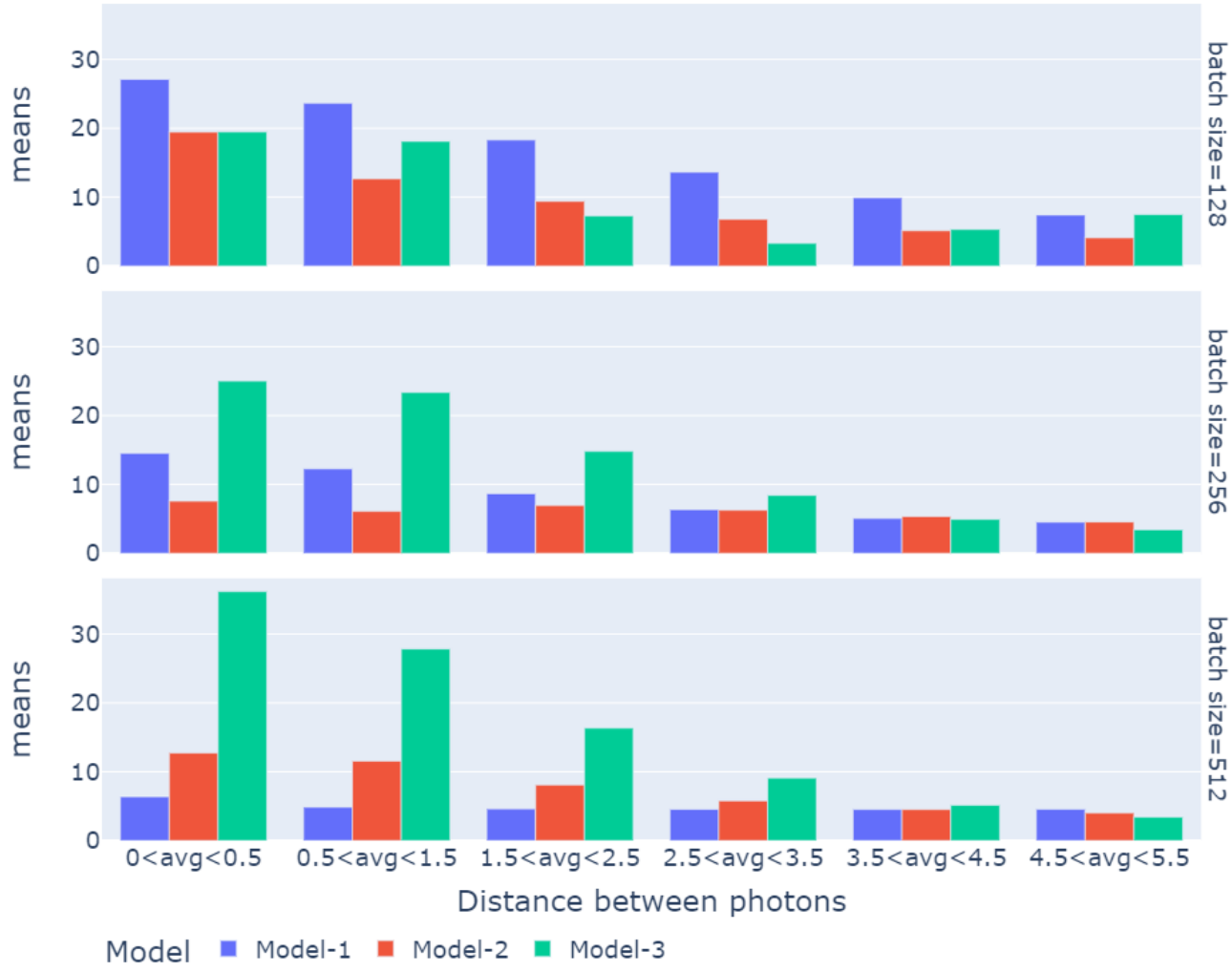
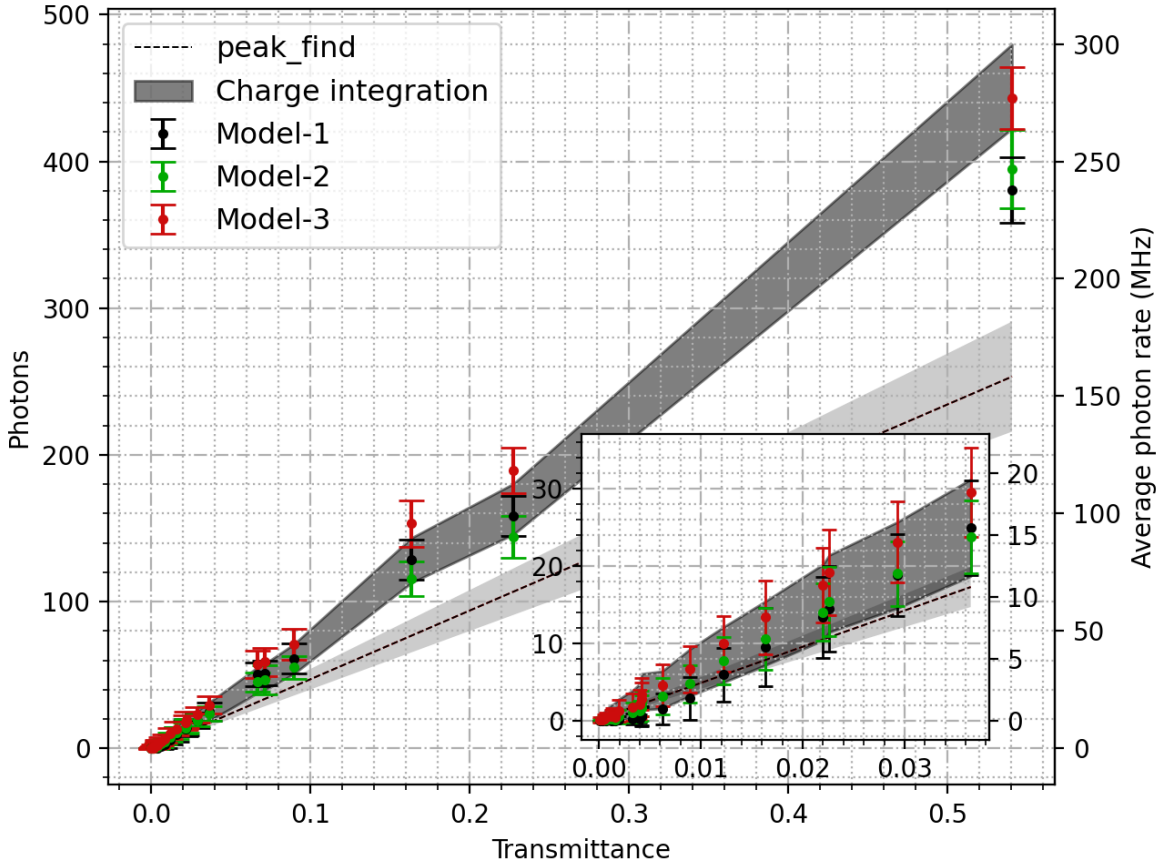
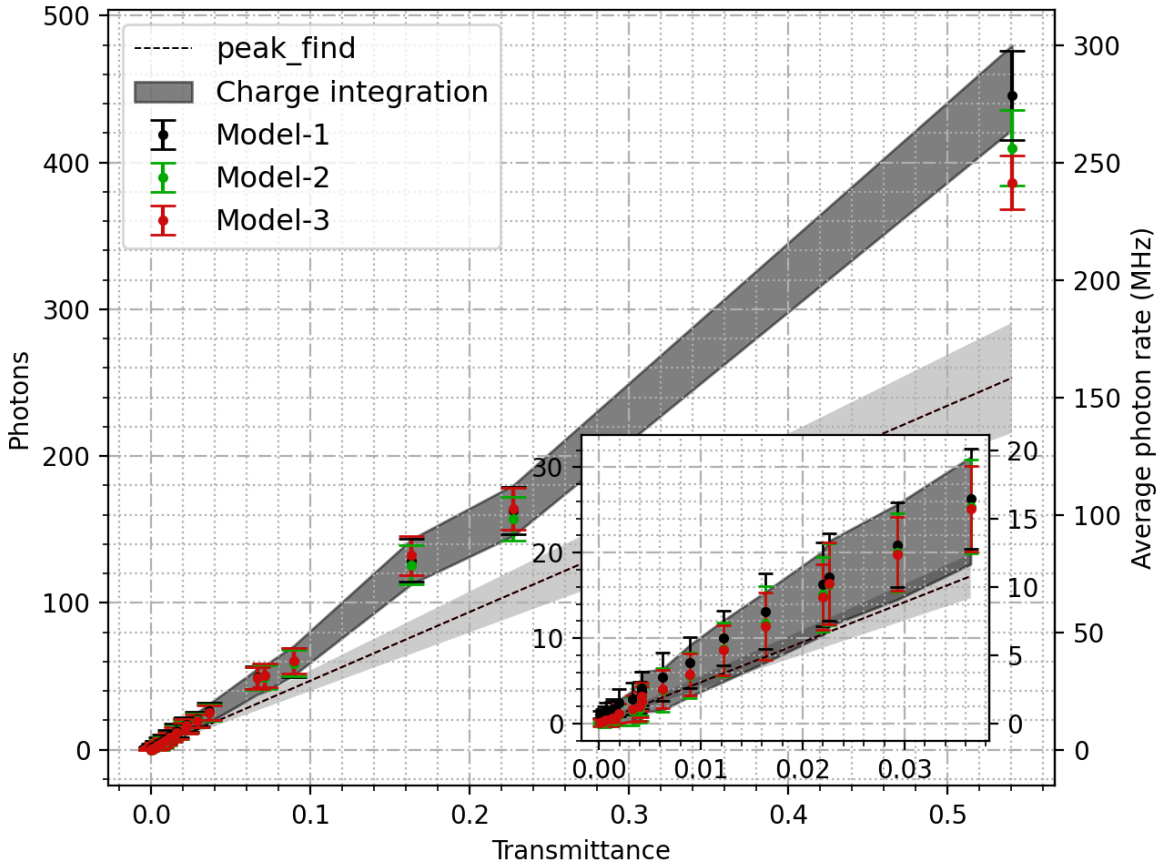


Figure 4.18: Evaluation comparison of models: Class1 is the average distance between successive photons, while class 2 indicates model used. Here we have used 2 CNN models with both configurations mentioned in Fig. 4.17, and one ResNet model, the shallower one. The plot-rows are for Mini-batch size configurations and y-axis of each 18 sub-plots are average of difference between model prediction and ground truth.

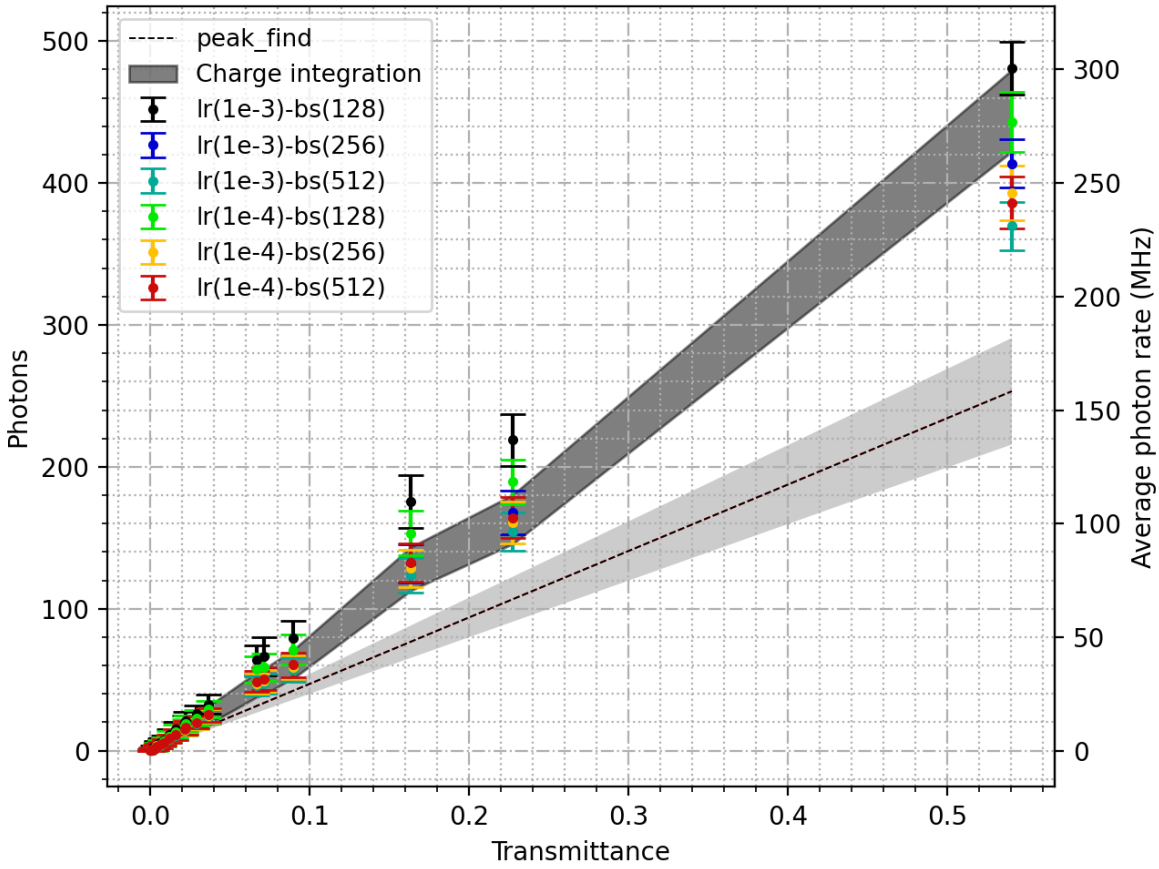


(a) Lab measurement model prediction comparison between two CNN model configurations and shallowest ResNet configuration. Here Model-1 has two Convolutional layer with filter size 4 and 2 while Model-2 has also 2 Convolutional layers but with filter size 4 and 32, while third model, ResNet is shallowest model among ResNet models. All models are trained with learning rate 10^{-4} and Mini-batch size 128.

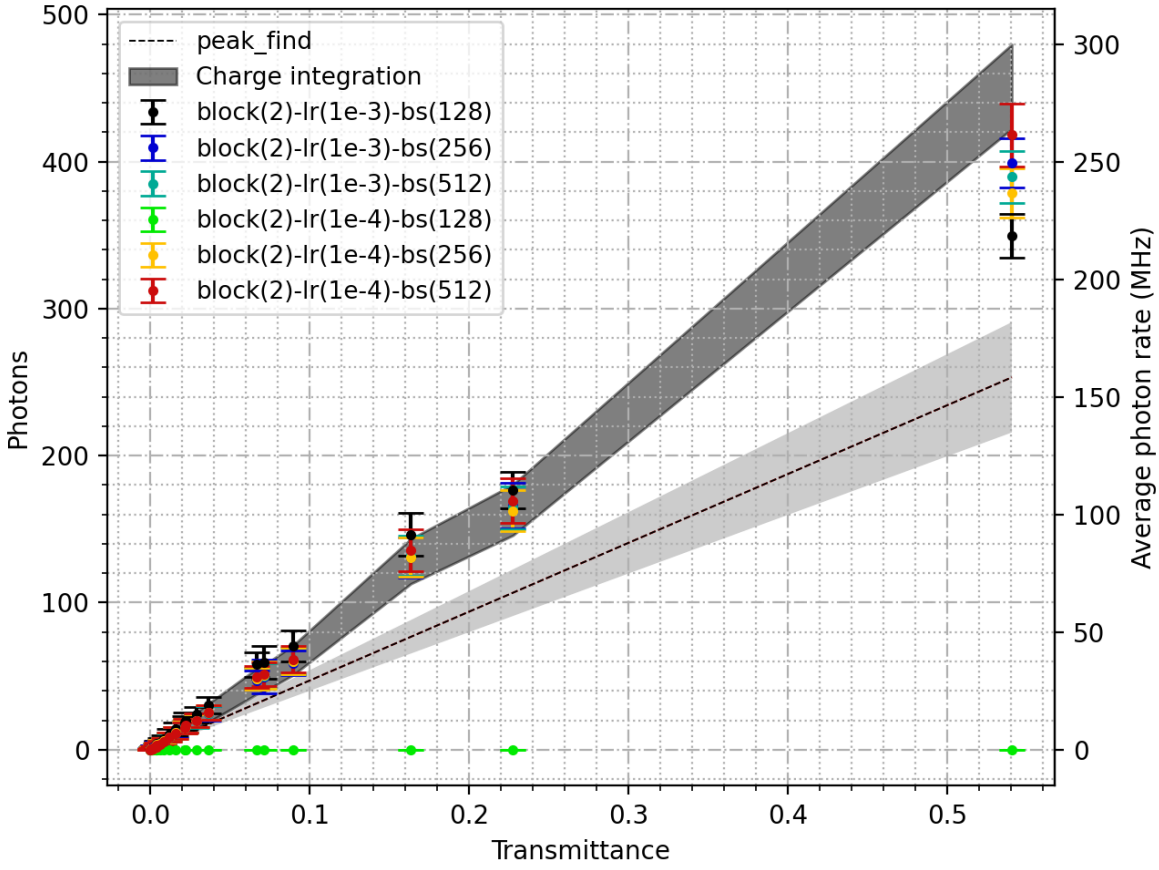


(b) Lab measurement model prediction comparison between two CNN model configurations and shallowest ResNet configuration. Here, models and in the figure are same as they are mentioned in Fig. 4.19a.

Figure 4.19: Model prediction comparison, between simple CNNs and ResNets

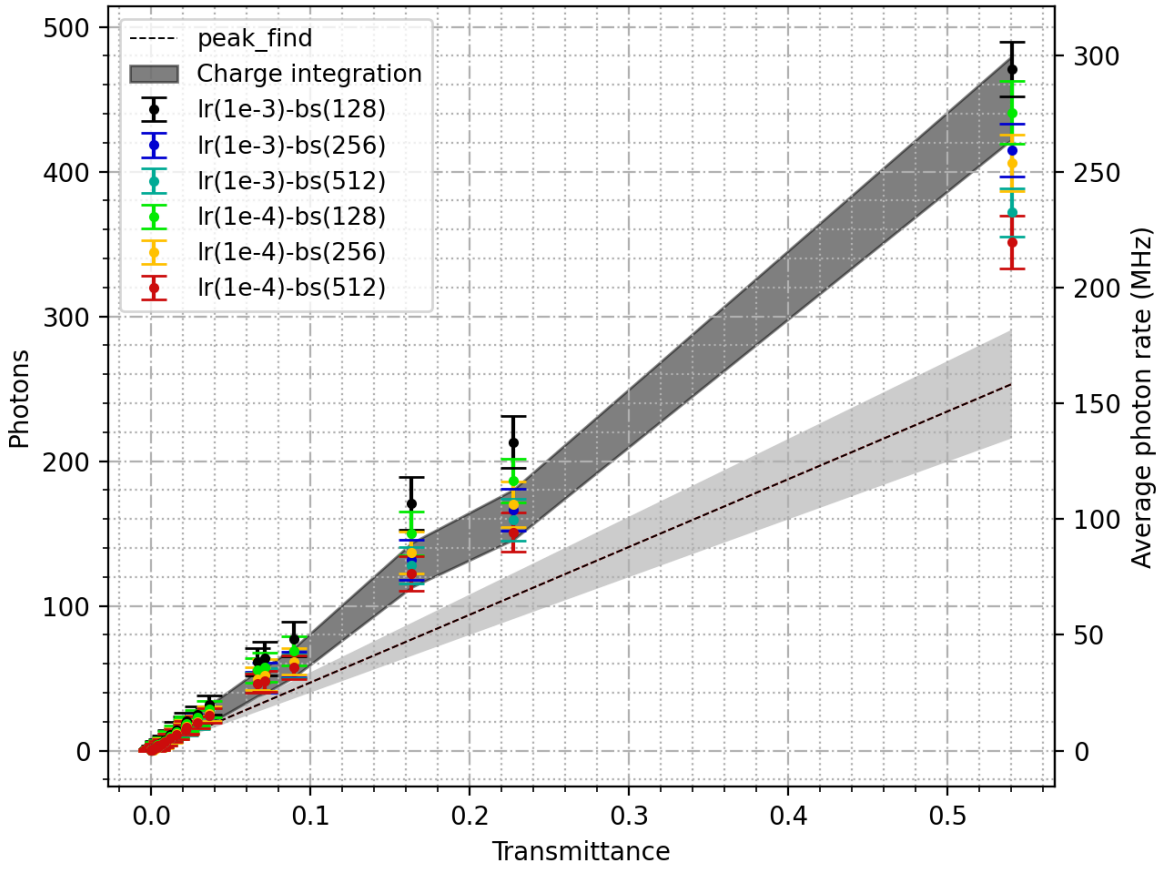


(a) Shallowest(with one block) ResNet architecture prediction comparison, with different configurations of learning rate and mini-batch size

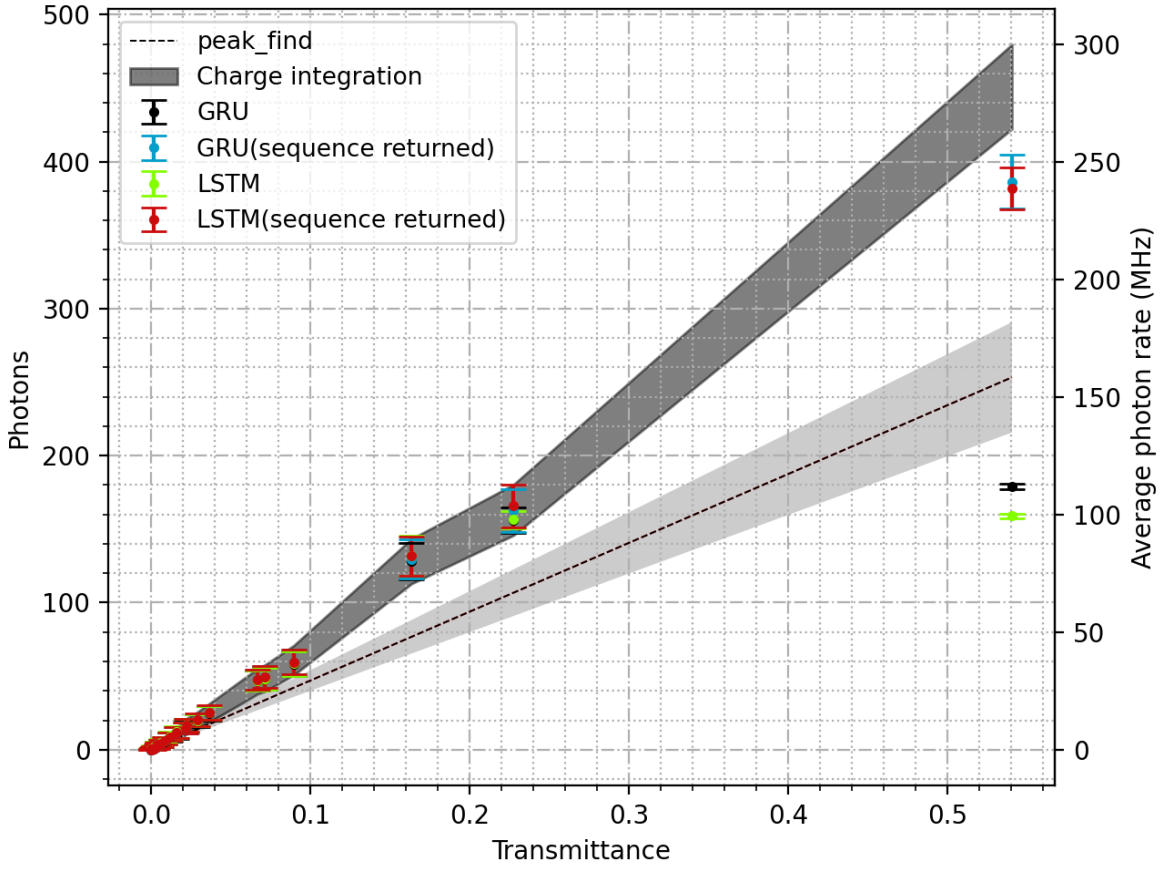


(b) Intermediate(with two blocks) deep ResNet architecture prediction comparison, with different configurations of learning rate and mini-batch size

Figure 4.20: Prediction comparison when different configurations are used with ResNet models

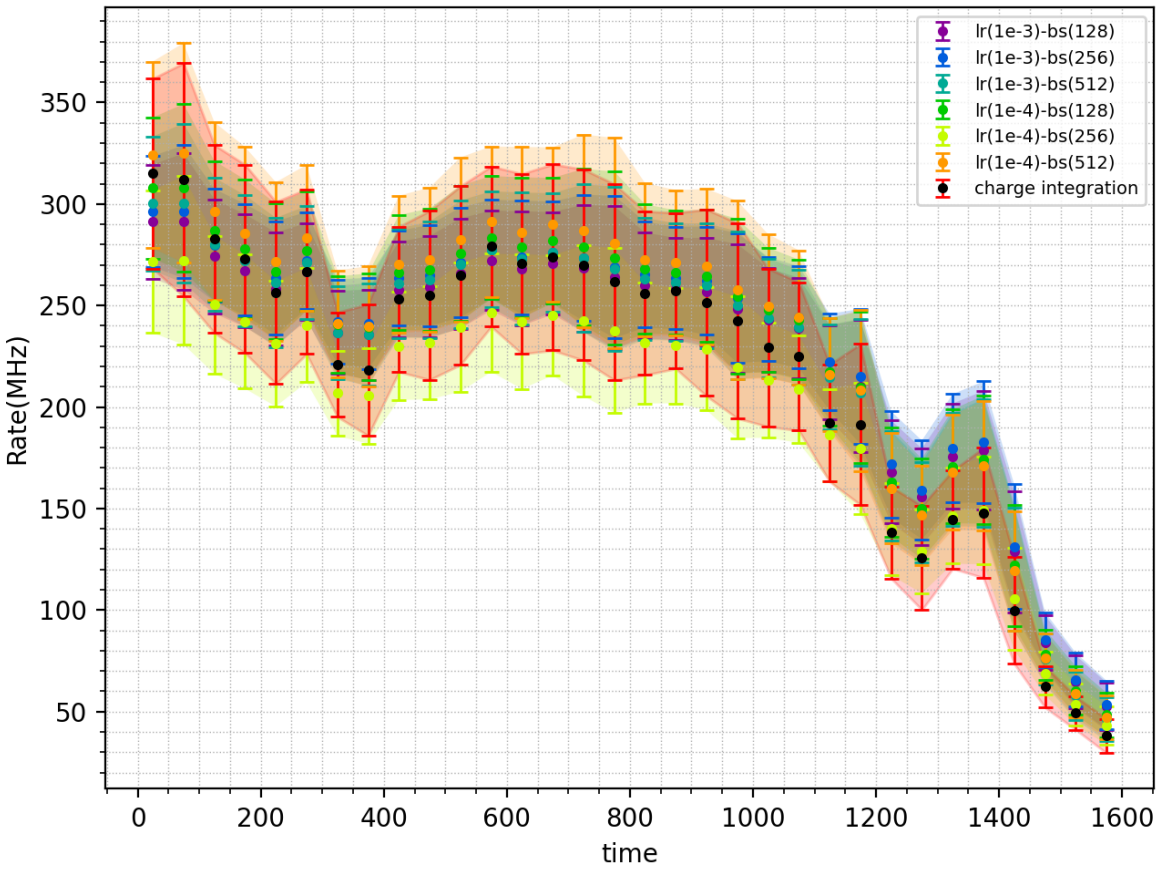


(a) Deeper(with three blocks) ResNet architecture prediction comparison, with different configurations of learning rate and mini-batch size

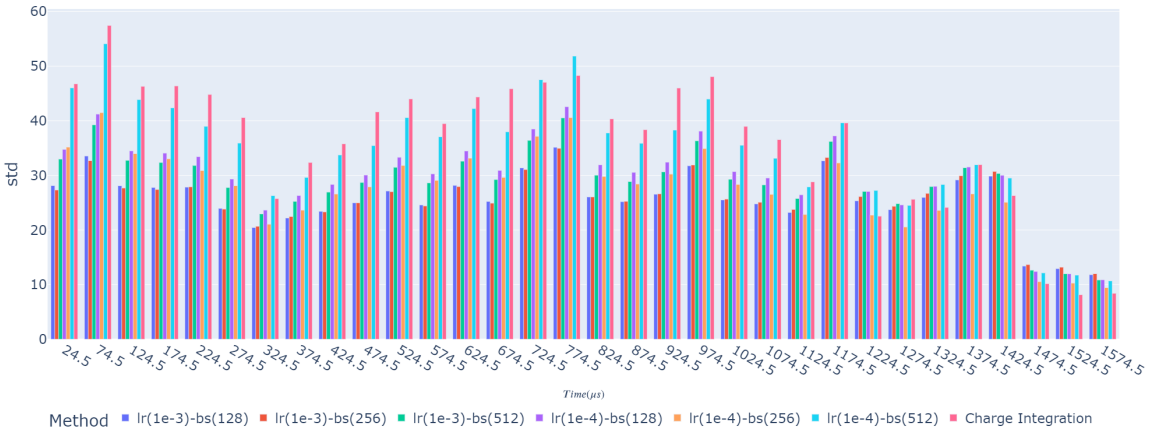


(b) Prediction comparison when LSTM and GRU are used with returned sequence, and when sequences were not returned

Figure 4.21: Prediction comparison of models

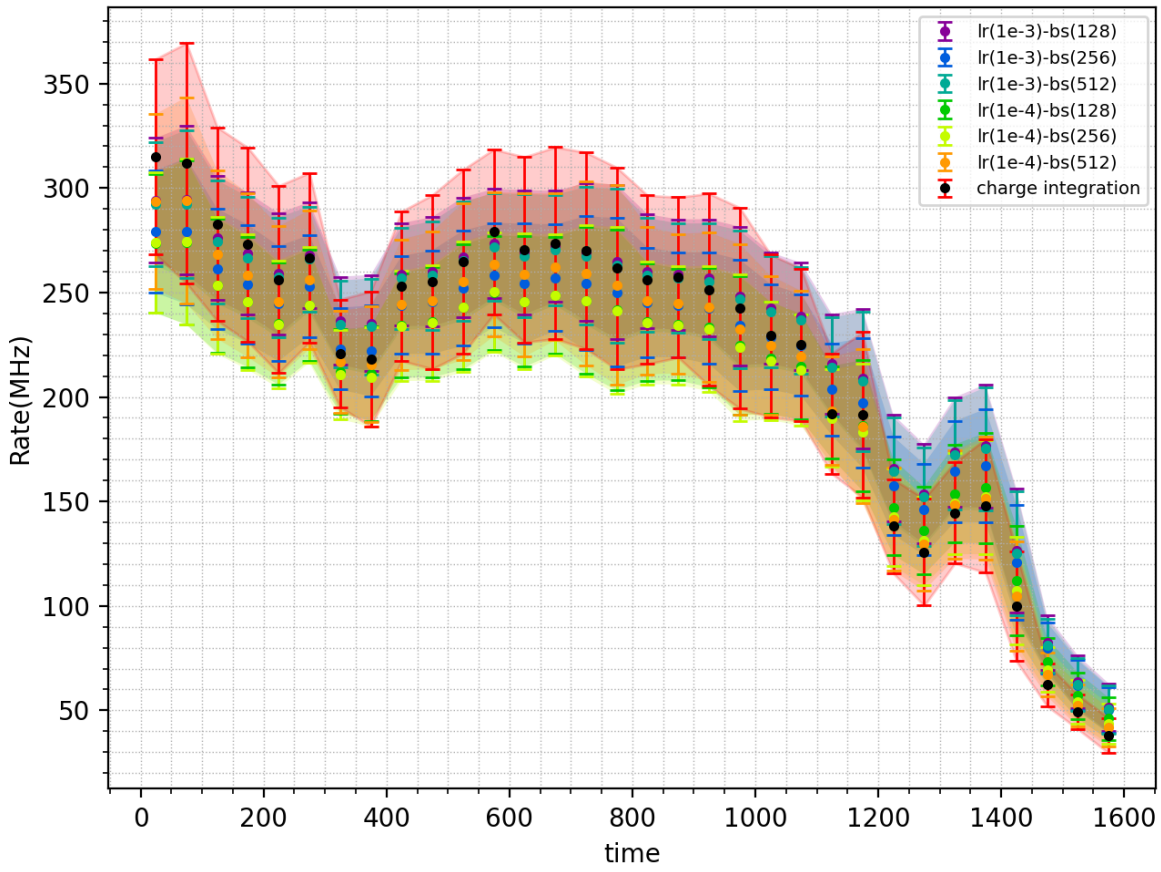


(a) Model predictions(μs) from Sirius when shallowest ResNet model is used.

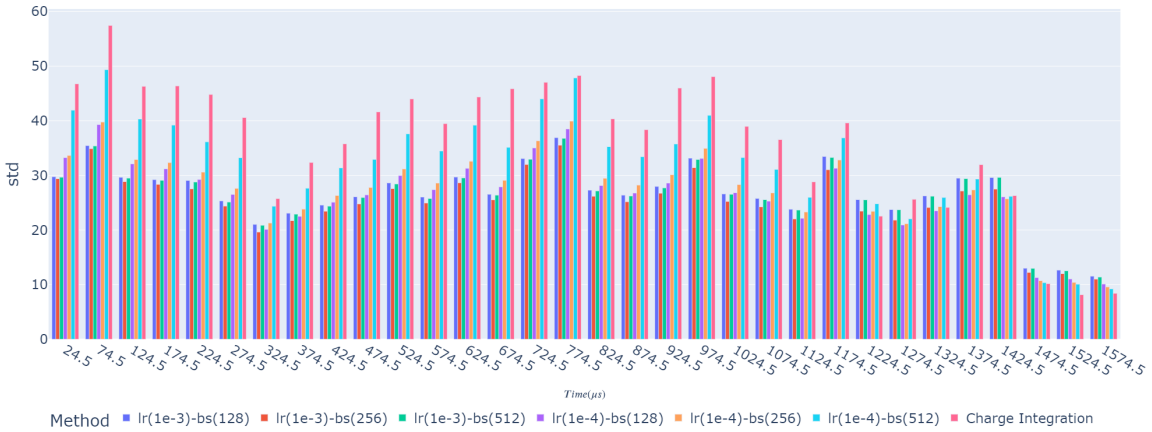


(b) ResNet-block-1

Figure 4.22: Shallower ResNet architecture with different configurations is used for training, and each error bar presents average and standard deviation over 50 samples form Sirius measurements(test dataset).

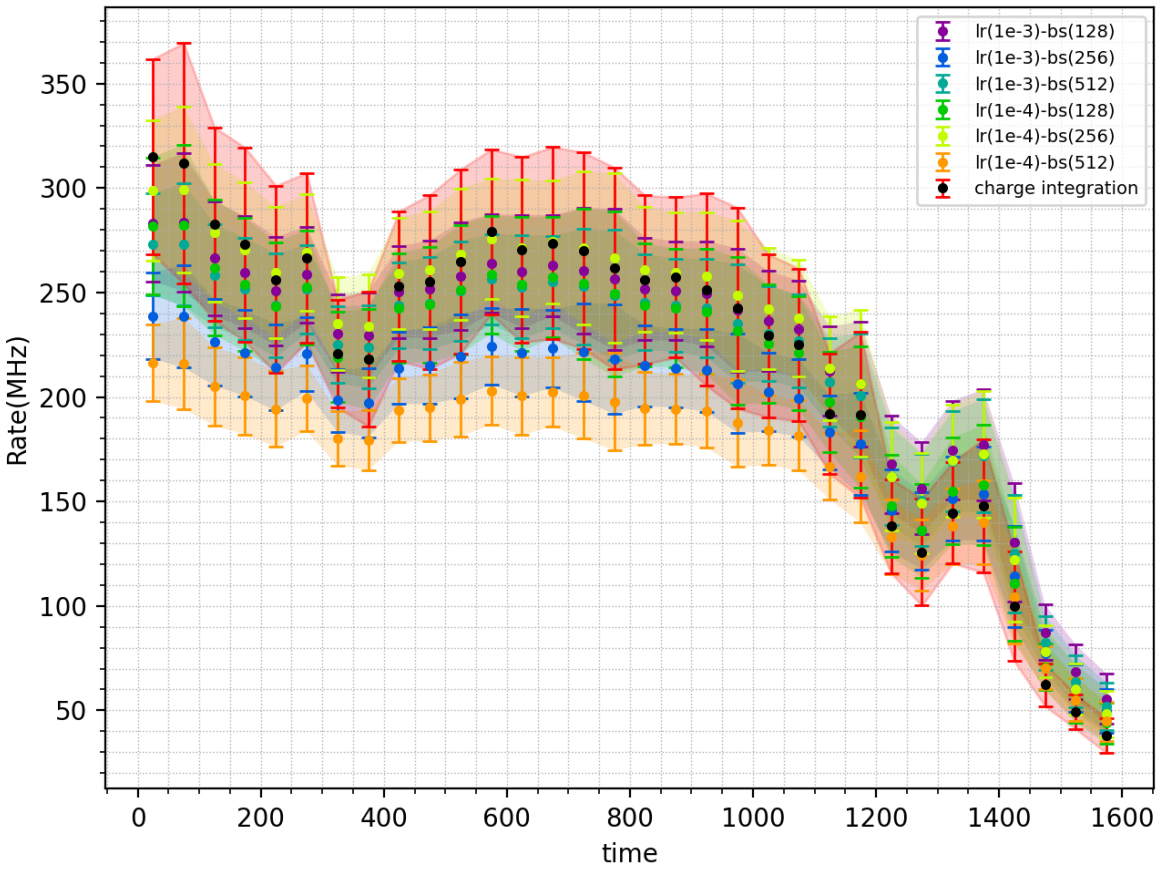


(a) Model predictions(μs) from Sirius when model slightly deeper ResNet model is used.

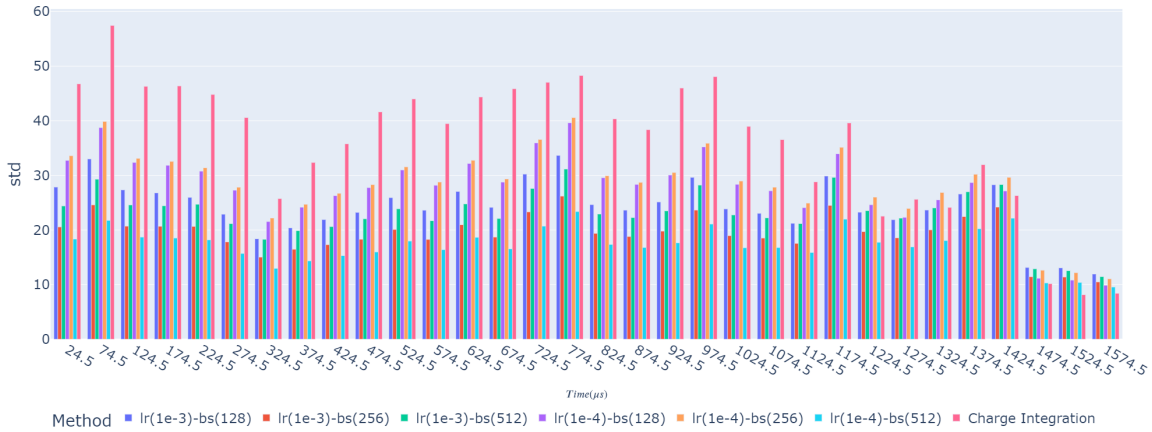


(b) ResNet-block-1

Figure 4.23: Intermediate deep ResNet architecture with different configurations is used for training, and each error bar presents average and standard deviation over 50 samples form Sirius measurements(test data set).

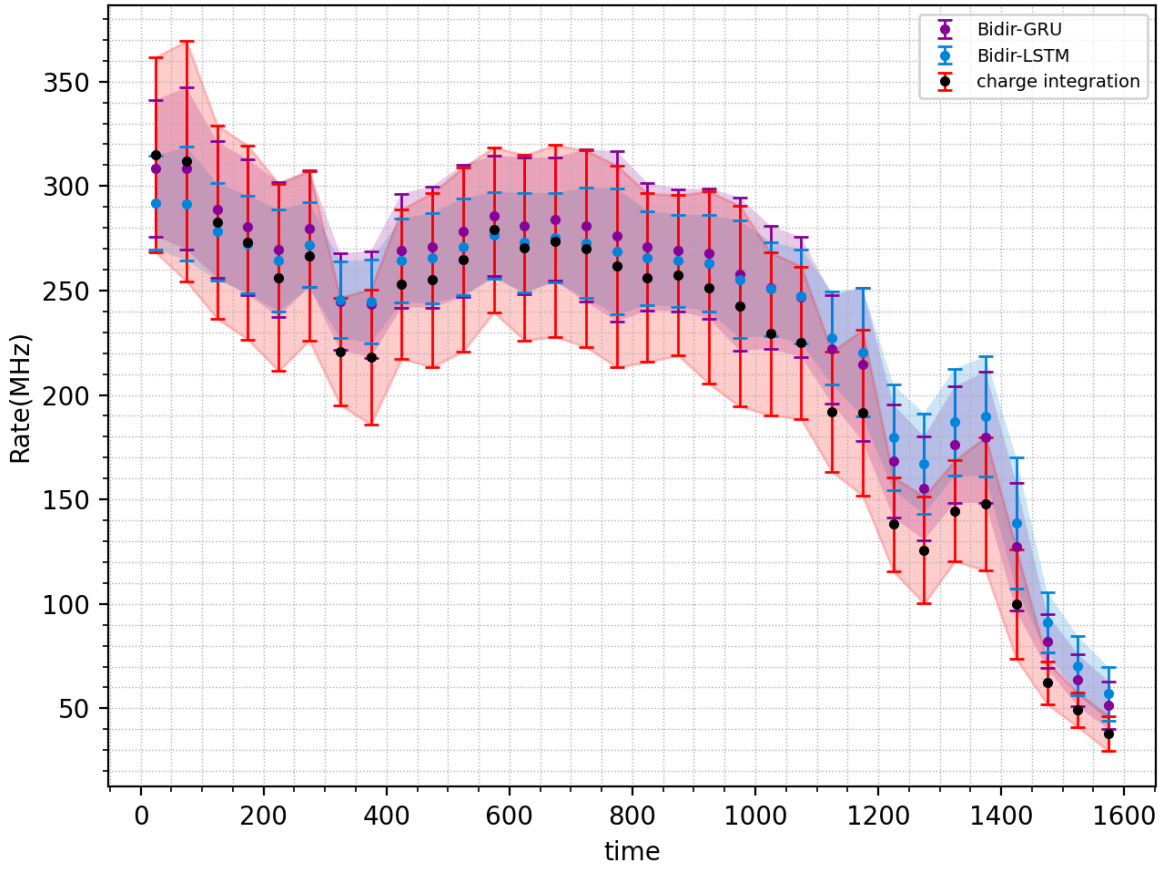


(a) Model predictions(μs) from Sirius when model deeper ResNet model is used.

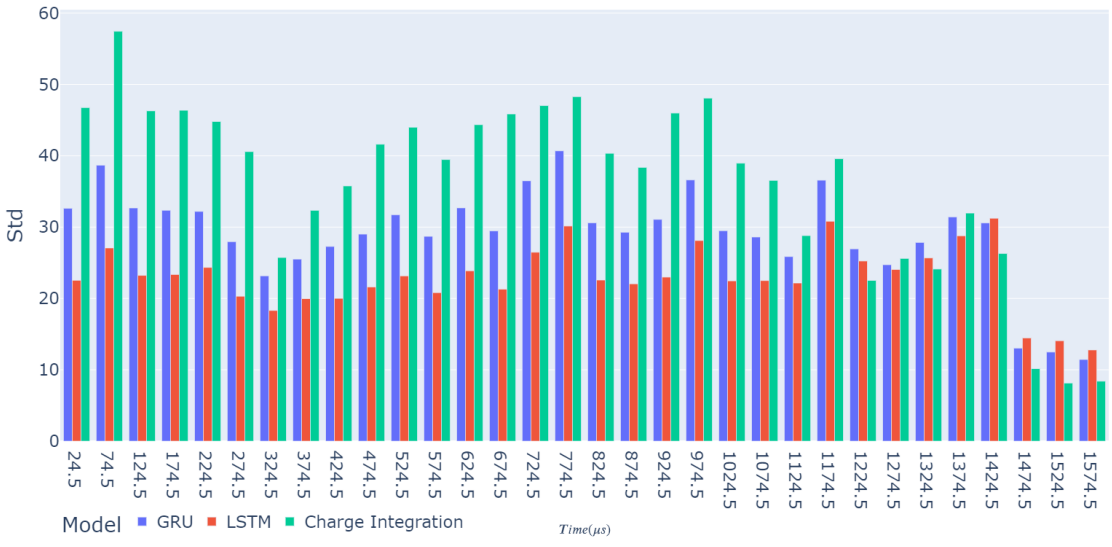


(b) ResNet-block-1

Figure 4.24: Intermediate deep ResNet architecture with different configurations is used for training, and each error bar presents average and standard deviation over 50 samples form Sirius measurements(test data set).



(a) Model predictions(μs) from Sirius when shallowest Bidirectional LSTM and GRU used with returned setuence manner model is used.



(b) ResNet-block-1

Figure 4.25: Bidirectional LSTM and GRU model prediction comparison, and each error bar presents average and standard deviation over 50 samples form Sirius measurements(test data set).

5

CONCLUSION

To conclude, first Fully connected Neural Network is used to determine prediction power of neural network at for low rate measurements, with different model parameters and hyper-parameters. From an important test, checking whether Stochastic Gradient-descent optimizer is better in our case or Adam optimizer, we determine that Adam optimizer is best selection for our task.

Then, by using Adam optimizer, at the same time other advanced neural network architectures, such as Convolutional Neural Networks, Long-Short Term Memory units and Gated Recurrent Unit, are used, which have shown improved results, as expected from references used, and also applied for low rate measurements from Sirius. Then, predictions of methods used are compared, which has proved that the Various neural network method prediction patterns are quite similar to Charge Integration. All the method predictions has showed some offset for background measurements from low rate Sirius measurements. Amongst, all the methods, Neural Network method predictions are little more stable. From loss curve, evaluation and prediction comparison, we found that Convolutional Neural network are providing best results.

In order to check, whether Neural Networks predictions are similar also for high rate measurements with larger sample size, in similar way we used Monte-Carlo simulated dataset for training, testing and evaluation, with GHz rate examples(GHz experiment), and used three different type of convolutional neural network architectures. Two of them are very simple but different in layer arrengement, and third one is Residual Network(ResNet). ResNet architecture is used with three different depth configurations. From CNN trials we found that for Simpler configurations are providing predictions closer to CHarge Integration method. For CNNs, all the best results are obtained using Adam optimizer with learning rate 0.0001, for lab measurements and Sirius measurements.

For low rate measurements we have found that, LSTMs and GRUs are working very well below 80MHz, and 0.1 transmittance, but beyond that limit, they are following peak find prediction. This is because output from each unitcell is passed to next unitcell via cell state, rather than using each output for rate determination, which made architecture less accurate. Therefore, in the second experiment(GHz experiment) we used returned sequences from each of the unit cell from LSTM and GRU models, and again compared them with Charge Integration, and found out that the use of independent output from each cell helps improving predictions while using LSTM or GRU.

5.1 DETERMINATION OF GROUND TRUTH

For the low rate measurements, Peak find method works very well, and for performance comparison, peak find predictions can be used as ground truth. And as explained above, Neural Network predictions and Charge Integration predictions at low rate are in agreement with Peak find predictions. But for high rate measurement, when signal starts overlapping, peak find can not be trusted. Therefore, for our comparison task there is no direct ground truth available. But, we tried to use indirect approach for ground truth. After confirming that NN predictions are similar for both Lab and Sirius measurements, we used evaluation of three model, where we used second type of evaluation dataset. From this evaluation, we determined certain order of three model evaluation with uncertainty. We then compared it with model prediction order over test data from laboratory experiment, from which we again found that overall more simpler the model, closer it is to the ground truth. However, analysis for much stronger evidence for ground truth determination can be one of the follow-up task of this master thesis.

BIBLIOGRAPHY

[1] F. G. Michelson A. A. ; Pease, 'Measurement of the diameter of orionis with the interferometer.', *Astrophysical Journal,*, vol. 53, pp. 249–259, 1921.

[2] E. W. Leonard Mandel, *Optical Coherence and Quantum Optics*. 1995, pp. 0–839, ISBN: 9781139644105.

[3] M. Ryle, 'The application of interferometric methods in radio astronomy', *Vistas in Astronomy*, vol. 1, pp. 532–541, 1955, ISSN: 0083-6656. DOI: https://doi.org/10.1016/0083-6656(55)90067-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0083665655900672.

[4] L. A. H. B. R. TWISS R., 'Correlation between photons, in coherent beams of light, detected by a coincidence counting technique', *Nature*, vol. 180, pp. 324–326, 1957, ISSN: 4581. DOI: https://doi.org/10.1038/180324a0.

[5] B. R. Hanbury and T. R. Q., 'Interferometry of the intensity fluctuations in light iv. a test of an intensity interferometer on sirius a', in *Proc. R. Soc. Lond. A*, 1958, 248222–237.

[6] F. R. HANBURY BROWN, *The Intensity Interferometer*. 1974, pp. 0–199, ISBN: 0 85066 072 6.

[7] L. Morgan B. L. ; Mandel, 'Measurement of photon bunching in a thermal light beam', *Physical Review Letters*, vol. 16, no. 22, pp. 1012–1015, 1966.

[8] *F-Praktikum experiment 45: Photonenstatistik*. 2017. [Online]. Available: http : / / www . fp . fkp . uni - erlangen . de / fortgeschrittenen - praktikum / versuchsangebot - fuer - bsclanf / BSc - Versuchsanleitungen/B45.pdf..

[9] M. Fox, *Quantum optics: an introduction*. 2007, pp. 16–19, ISBN: 978-0-19-856673-1.

[10] A. Zmija, P. Deiml, D. Malyshev, A. Zink, G. Anton, T. Michel and S. Funk, 'Led as laboratory test source for astronomical intensity interferometry', *Opt. Express*, vol. 28, no. 4, pp. 5248–5256, 2020. DOI: 10.1364/OE.28.005248. [Online]. Available: http://www.osapublishing.org/oe/abstract.cfm?URI=oe-28-4-5248.

[11] R. Loudon, *The quantum theory of light (*. 2017, ISBN: 9780198501763. [Online]. Available: http://rplab.ru/~as/2000%20-%20R.Loudon%20-%20The%20Quantum%20Theory%20of%20Light%20-%203rd%20ed%20Oxford%20Science%20Publications.pdf.

[12] H. Donald, *Book*, ser. A textbook of psychology. 1949, pp. 1–298, ISBN: 978-0-7872-1103-5. [Online]. Available: https://archive.org/details/textbookofpsycho00hebb/mode/2up.

[13] R. Frank, *Book*, ser. Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. 1962, pp. 1–626, ISBN: Spartan Books. [Online]. Available: https://safari.ethz.ch/digitaltechnik/spring2019/lib/exe/fetch.php?media=neurodynamics1962rosenblatt.pdf.

[14] W. B, *Report*, ser. An adaptive "ADALINE" neuron using chemical "memistors". 1960, pp. 0–28, ISBN: 1553-2. [Online]. Available: https://isl.stanford.edu/~widrow/papers/t1960anadaptive.pdf.

[15] M. A. Widrow Bernard; Lehr, 'Computer typesetting of technical journals on UNIX', in *Proceedings of the IEEE: 1990: 30 years of adaptive neural networks: perceptron, madaline, and backpropagation National Computer Conference*, 1990, pp. 1415–1442.

[16] E. R. James A. Anderson, *Book*, ser. Talking Nets: An Oral History of Neural Networks. 2000, pp. 1–358, ISBN: 9780262511117.

[17] D. S, 'The numerical solution of variational problems.', *Journal of Mathematical Analysis andApplications,*, vol. 5, no. 1, pp. 30–45, 1962.

[18] K. J. Henry, 'Gradient theory of optimal paths', *ARS Journal*, 1960.

[19] J Schmidhuber, 'Deep Learning in Neural Networks: An Overview:' *ArXiv e-prints*, 2014. arXiv: 404.7828v4 [cs.NE].

[20] Y. H. G. LeCun Yann; Bengio, 'Deep learning', *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[21] T. Hubel DH; Wiesel, 'Receptive fields of single neurones in the cat's striate cortex', *J. Physiol*, vol. 148, no. 3, pp. 574–91, 1959.

[22] ——, 'Cresceptron: A self-organizing neural network which grows adaptively', *n International Joint Conference on Neural Networks (IJCNN), v*, vol. 1, pp. 576–581, 1992.

[23] A. N. Weng J. J. and T. S Huang, 'Learning recognition and segmentation using the cresceptron.international journal of computer vision', *International Joint Conference on Neural Networks (IJCNN), v*, vol. 25, no. 2, pp. 109–143, 1997.

[24] H. F. B. Y. Ranzato M. A. and Y LeCun, 'Unsupervised learning of invariant feature hier-archies with applications to object recognition', in *Proceedings of the Computer Vision and Pattern Recognition Conference(CVPR'07)*, IEEE Press, 2007, pp. 1–8.

[25] M. A. Scherer D. and S Behnke, 'Evaluation of pooling operations in convolutional architecturesfor object recognition.', in *Proc. International Conference on Artificial Neural Networks(ICANN)*, 2010, pp. 92–101.

[26] D. Eck and J. Schmidhube, '
learning the long-term structure of the blues"', in *Artificial NeuralNetworks — ICANN 2002. Berlin, Heidelberg:*, Springer Berlin Heidelberg, 2002, pp. 284–289.

[27] S. Hochreiter and J. Schmidhuber, '
long short-term memory".', in *Neural computation 9.8 (1997)*, 1997, pp. 1735–1780.

[28] A. C. Ian Goodfellow Yoshua Bengio, *Deep Learning*. 2016, pp. 0–839, ISBN: 978-0262035613.

[29] A. Maier, *Deep Learning lectures*. 2019, ISBN: University of Erlangen-Nurnberg.

[30] B. T Polyak, 'Some methods of speeding up the convergence of iteration methods', *Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964.

[31] J. L. B. Diederik P. Kingma, ' Adam : A method for stochastic optimization.', *ArXiv e-prints*, Jan. 2015. arXiv: 1412.6980v9 [cs.NE].

[32] Y. S. John Duchi Elad Hazan, 'Adaptive subgradient methods for online learning and stochastic optimization', *Journal of Machine Learning Research*, vol. 12, no. 5, pp. 2121–2159, 2011. [Online]. Available: https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf.

[33] M. S. N. S. B. R. Ashia C. Wilson Rebecca Roelofs, ' The Marginal Value of Adaptive Gradient Methods in Machine Learning', *ArXiv e-prints*, May 2017. arXiv: 1705.08292v2 [cs.NE].

[34] R. S. Nitish Shirish Keskar, ' Improving Generalization Performance by Switching from Adam to SGD', *ArXiv e-prints*, Dec. 2017. arXiv: 1712.07628v1 [cs.NE].

[35] M. Ohi, 'Illustrated guide to lstm's and gru's: A step by step explanation', 2018. [Online]. Available: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

[36] H. P. K.K., 'Photomultiplier tubes - basics and appli- cation.', 2007. [Online]. Available: https://www.hamamatsu.com/resources/pdf/etd/PMT_handbook_v3aE.pdf.

[37] ——, 'Photomultiplier tubes - construction and op- erating charac- teristics connections to external circuits.', 1998. [Online]. Available: https://wwwmu.mpp.mpg.de/docs/pmtconstruct.pdf..

[38] 'Fast. ta1000b-x fast, very low noise pulse / timing preamplifier user manual,' *Accessed: 12.10.2020.*, 2016.

[39] 'Fast. ta1000b-10/-50/-100/-200', *Accessed: 12.10.2020.*, 2016.

[40] 'Spectrum instrumentation. m4i.22xx-x8 - 8 bit digitizer up to 5 gs/s,' *Accessed: 13.10.2020*, 2020.

[41] M. Ohi, 'Calibration of photomultiplier tubes for intensity interferometry at h.e.s.s.', 2020. [Online]. Available: https://ecap.nat.fau.de/wp-content/uploads/2021/01/2020-MA-NVogel.pdf.