Western Graduate&PostdoctoralStudies

Western University
Scholarship@Western

Electronic Thesis and Dissertation Repository

6-12-2024 3:45 PM

UTILIZING MACHINE LEARNING TECHNIQUES FOR DISPERSION MEASURE ESTIMATION IN FAST RADIO BURSTS STUDIES

Hosein Rajabi, *The University of Western Ontario*

Supervisor: Daley, Mark, *The University of Western Ontario*
Joint Supervisor: Houde, Martin, *The University of Western Ontario*
A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science
© Hosein Rajabi 2024

Follow this and additional works at: https://ir.lib.uwo.ca/etd

Recommended Citation
Rajabi, Hosein, "UTILIZING MACHINE LEARNING TECHNIQUES FOR DISPERSION MEASURE ESTIMATION IN FAST RADIO BURSTS STUDIES" (2024). *Electronic Thesis and Dissertation Repository*. 10127.
https://ir.lib.uwo.ca/etd/10127

This Dissertation/Thesis is brought to you for free and open access by Scholarship@Western. It has been accepted for inclusion in Electronic Thesis and Dissertation Repository by an authorized administrator of Scholarship@Western. For more information, please contact wlswadmin@uwo.ca.

Abstract

Fast Radio Bursts (FRBs) are highly energetic, millisecond-duration bursts of energy often detected from extragalactic sources. Almost two decades after their initial discovery, the sources and underlying physical mechanisms of FRBs remain unknown. As FRB signals propagate through space, they interact with matter, resulting in a temporal dispersion with frequency. The amount of dispersion is quantified through the Dispersion Measure (DM), which specifies the time delay before the arrival of a pulse as a function of frequency. The DM plays a crucial role in estimating the distance to the FRB source and can reveal information about the physical conditions at the source. Furthermore, accurate DM estimations have far-reaching implications beyond the direct study of FRBs. They can significantly impact other areas of astrophysics, such as the study of the intergalactic medium, the mapping of cosmic structures, and providing insights into the fundamental laws of physics. These aspects highlight the importance of precise DM measurements in advancing our understanding of the universe. However, the accurate estimate of the DM is often complicated and subject to human error. Automating and improving DM estimations has become an even more important task with the increasing number of detected FRBs and the commissioning of new FRB experiments. In this thesis, we explore various deep-learning models for estimating FRB DMs. To enhance the results of training models, we create a set of simulated FRB data, as real FRB data are often noisy, to develop extensive training sets. We discuss the outcomes of different deep learning models and identify the most promising ones. This research is crucial for understanding and analyzing future FRB data, which are expected to grow exponentially by the commissioning of new experiments and facilities such as the Square Kilometer Array (SKA).

Keywords: Fast Radio Bursts, Dispersion Measure Estimation, Machine Learning

Lay Summary

Fast Radio Bursts (FRBs) are brief, intense radiation flashes from distant galaxies, lasting typically milliseconds. Discovered nearly two decades ago, their origins and mechanisms remain unknown. Traveling vast distances, these bursts interact with matter, affecting their arrival times at our telescopes. This interaction is measured by the Dispersion Measure (DM), which helps determine the travel distance and conditions encountered. Measuring the DM is essential not only for studying FRBs but also for insights into the universe's structure and fundamental physics. Accurately measuring the DM is, however, challenging. This thesis focuses on using advanced machine learning models to improve DM estimations. By training models on simulated data, we aim to develop more accurate methods for analyzing FRB signals. This is crucial for handling the surge in data from upcoming astronomical facilities like the Square Kilometer Array (SKA), poised to enhance our understanding of the universe.

Acknowledgements

I would like to express my profound gratitude to my thesis supervisors, Prof. Martin Houde from the Department of Physics and Astronomy, and Prof. Mark Daley from the Department of Computer Science. Their invaluable guidance and expertise have been fundamental to my development and understanding in this interdisciplinary research, which bridges the complex realms of astronomy and machine learning. Prof. Houde patiently taught me many aspects of radio transients, their physics, and the properties of their signals, all of which were pivotal to the development of this research. Prof. Daley's critical insights into the machine learning aspects of my project were particularly enriching, enhancing the methodology and execution of my research. I am deeply thankful for the opportunity they provided me to explore these fascinating fields, and their mentorship has been a cornerstone of my academic journey.

I am also thankful to all the staff in the main offices of both the Department of Physics and Astronomy and the Department of Computer Science. Their administrative support was essential and greatly appreciated. Additionally, I would like to acknowledge my Master's Advisory Committee for their advice.

Special thanks are due to Dr. Mohammed Chamma from the Department of Physics and Astronomy for his helpful discussions on handling Fast Radio Burst data and the conventional dispersion measure estimation methods.

I greatly value the friendship and support of my Computer Science friends, Nima, Ghazal, and Alireza, whose discussions about graduate student life have been both enlightening and comforting.

I am fortunate to have worked alongside some brilliant group mates who have contributed to a stimulating and productive research environment. I am particularly grateful to Aishwarya Kumar, Katie Brown, and Ningyan (Louis) Fang, as well as all members of Prof. Daley's group in Computer Science, for their collaboration and shared enthusiasm.

A big thank you to my family for their unwavering support and love. To my mom and dad, your encouragement has been the foundation of my hope and strength. Special thanks to my sisters, Fereshteh and Farzaneh, and also to my brothers-in-law, Javad and Mehrdad, for their support and encouragement. To Sanaz, my partner and long-time friend, thank you for all your support and patience through the challenges and triumphs.

Finally, a delightful thank you to my nephew Alex, whose stories about his imaginary friend Mr. Mouse have brought fun and laughter into my days, reminding me of the joy in the simple things.

Thank you all for your support and encouragement, which have been pivotal to my academic journey.

Contents

Abstract ii

Lay Summary iii

Acknowledgements iv

Table of Contents v

List of Tables vii

List of Figures viii

List of Abbreviations ix

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Thesis Outline . . . 4

2 Background 5
- 2.1 Exploring FRB Characteristics . . . 5
  - 2.1.1 Dispersion Measure (**DM**) . . . 5
  - 2.1.2 Scattering . . . 6
  - 2.1.3 Scintillation . . . 7
  - 2.1.4 Pulse Width of an FRB . . . 7
  - 2.1.5 The sub-burst slope law . . . 8
- 2.2 Machine learning (ML) . . . 9
  - 2.2.1 Tree-Based Models . . . 10
  - 2.2.2 Support Vector Machines . . . 12
  - 2.2.3 Clustering Algorithms . . . 13
  - 2.2.4 Dimensionality Reduction Techniques . . . 13
- 2.3 Artificial Neural Networks . . . 14
  - 2.3.1 Multi Layer Perceptron . . . 15
  - 2.3.2 Activation Functions . . . 16
  - 2.3.3 Optimizers . . . 17
- 2.4 Convolutional Neural Networks . . . 20
  - 2.4.1 Convolutional Layer . . . 21
  - 2.4.2 Pooling Layer . . . 22

2.4.3 Dropout 23
2.4.4 Batch Normalization 24
2.5 Residual Neural Network 26
2.6 Vision Transformer (ViT) 27

3 Related Work 30

4 Methodology 34
4.1 Data 34
4.1.1 Data Generation 34
4.2 Model Training 40
4.2.1 Baseline CNN Model 40
4.2.2 DM Estimation utilizing Transfer Learning 42
4.3 Model Evaluation 44

5 Results & Discussion 45
5.1 Dataset 45
5.2 Baseline CNN Model 45
5.3 Vision Transformers 47
5.4 Resnet50 49
5.5 Discussion 51

6 Conclusion and Future Work 53
6.1 Conclusion 53
6.2 Future Work 53

Bibliography 55

Curriculum Vitae 61

List of Tables

3.1 Average F2 scores and standard deviations of supervised models trained for classifying FRBs as repeaters and non-repeaters [34]. 32
3.2 Performance of unsupervised algorithms trained for classifying FRBs as repeaters and non-repeaters [65]. 33

4.1 Number of Reported FRB Sources through various telescopes according to FRBSTATS [53]. 35
4.2 Telescope spectral information for data generation 36
4.3 Architecture of Baseline CNN Model. 41

5.1 Results of the baseline CNN Model over five runs 46
5.2 Results of the ViT Model over five runs 47
5.3 Results of the Resnet50 Model over five runs 50

List of Figures

1.1 Dispersed Arrival of FRB 110220 Signal 2
1.2 Left: Two dedispersed FRBs from the source FRB121102A. Right: Verification of the sub-burst slope law over nine distinct FRB sources 4

2.1 The framework of the TRDM of Rajabi et al. (2020) [46] 8
2.2 Illustration of a basic decision tree model to determine the tastiness of papayas [50]. 11
2.3 Illustration of a random forest model [29]. 12
2.4 Illustration of an MLP with two hidden layers and two outputs [30]. 16
2.5 Illustration of a simple two-dimensional convolution Operation [43]. 21
2.6 Illustration of max pooling and average pooling with their respective results [60]. 22
2.7 Illustration of data fellow through various layers—convolutional, pooling, and fully connected—to produce final results in CNNs [26]. 23
2.8 Illustration from the original Dropout publication [54] showcasing the concept . 24
2.9 Illustration of a residual block from the original paper [25]. 26
2.10 ViT architecture. Taken from [18]. 28

3.1 Illustration of the dual-path network architecture by Agarwal et al. [2] employing both frequency-time and DM-time images as inputs 31

4.1 Illustration of randomly generated background noise 37
4.2 Illustration of the convolution process between a Gaussian profile and a scattering profile 38
4.3 Illustration of the effect of scintillation applied to a signal. 39
4.4 Illustration of the generated data 40
4.5 Illustration of Resnet50 architecture 43

5.1 Illustration of DM distribution for training and validation sets 46
5.2 Histogram of absolute differences for DM predictions by the baseline CNN model. 47
5.3 Performance visualization of the fine-tuned ViT model 48
5.4 Histogram of residuals for DM predictions by the ViT model. 49
5.5 Performance visualization of the fine-tuned ResNet50 model 50
5.6 Histogram of Residuals for DM Predictions.. 51

List of Abbreviations

ASKAP	Australian Square Kilometre Array Pathfinder
BN	Batch Normalization
CHIME	Canadian Hydrogen Intensity Mapping Experiment
DM	Dispersion Measure
DSA	Deep Synoptic Array
FAST	Five-hundred-meter Aperture Spherical Telescope
FC	Fully Connected
FN	False Negative
FP	False Positive
FRB	Fast Radio Burst
GBT	Green Bank Telescope
GMRT	Giant Metrewave Radio Telescope
LOFAR	Low Frequency Array
LPA	Large Phased Array
MAE	Mean Absolute Error
ML	Machine Learning
MLP	Multi-layer Perceptron
MSE	Mean Squared Error
pc	Parsec (1 pc $\simeq$ 3.26 light years)
PCA	Principal Component Analysis
RFI	Radio Frequency Interference

RMSE	Root Mean Squared Error
SGD	Stochastic Gradient Descent
SRT	Sardinia Radio Telescope
SVM	Support Vector Machine
TLU	Threshold Logic Unit
TN	True Negative
TP	True Positive
TRDM	Triggered Relativistic Dynamical Mode
UMAP	Uniform Manifold Approximation and Projection
ViT	Vision Transformer
VLA	Very Large Array

Chapter 1

Introduction

Fast radio bursts (FRBs) are highly energetic bursts of energy detected at radio frequencies, ranging from 100 MHz to 8 GHz [45, 21]. The typical duration of FRBs is on the order of a few milliseconds (ms); however, durations as long as 3 seconds [13] and as short as a few microseconds [52] have also been reported. The energy of FRBs typically varies from 10^{35} erg to 10^{46} erg (1 erg = 10^{-7} J) [63]. To put this into perspective, the Sun releases the same amount of energy over a few years as an FRB does in just a few milliseconds.

The majority of FRBs are one-off events, with some repeating. These repeating FRBs provide great opportunities for follow-up observations and more in-depth studies. However, since their first discovery almost two decades ago, the origin and underlying physical mechanisms of FRBs have remained unknown despite intense and sustained theoretical [63] and observational efforts.

A key characteristic of FRBs is their broadband signal, which typically extends over an average bandwidth of nearly 200 MHz at 1.4 GHz. With the exception of one potential FRB originating from a galactic magnetar (SGR 1935+2154) [17], these bursts are predominantly recognized as extragalactic phenomena. The signal from an FRB source travels vast distances before being detected by radio telescopes on Earth. During this journey, the signal encounters ionized matter, leading to dispersion and scattering effects that vary with frequency. Notably, lower frequencies are more susceptible to these effects. For instance, an FRB that lasts only 1 ms at the source can be extended to several seconds upon detection due to these propagation effects, as illustrated in Fig. 1.1. The dispersion measure (DM), which quantifies the dispersion of an FRB signal, provides insights into the signal's travel history and the types of environments it has traversed. Furthermore, DM is instrumental in estimating the distance from Earth to the FRB source. Crucially, distinguishing these propagation effects from the actual emission signal is essential for revealing the true underlying physical mechanisms of FRBs. This separation aids theoretical investigations into the conditions at the source and helps elucidate the formation processes of FRBs.

Dedispersing recorded radio signals is a standard practice in radio astronomy, particularly when dealing with short pulses like those from pulsars or FRBs. This technique serves not only to distinguish genuine FRB signals from man-made radio interference, commonly referred to as Radio Frequency Interference (RFI), but also provides valuable estimates of the radio signal at the time of emission. The dedispersion process is currently performed using so-called coherent and incoherent techniques [5, 49, 62] as part of the FRB data cleansing and processing.

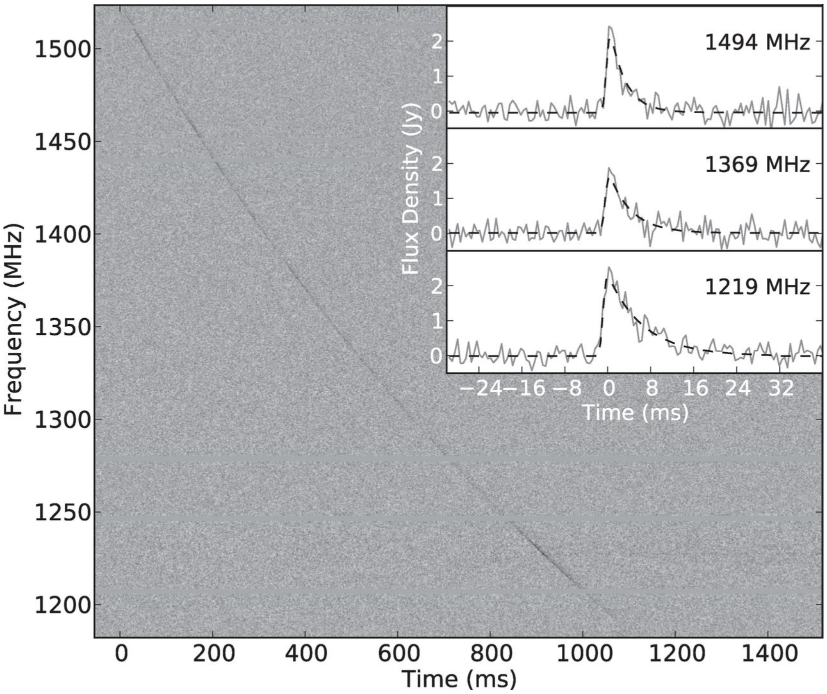

Figure 1.1: FRB 110220, as detected, is seen to stretch over approximately 1 s, while the dedispersion at DM = 944.38 cm^{-3} pc shows the signal to last on the order of only a few ms (see inset, in frequency bands of 25 MHz). FRB 110220 is estimated to be located at a distance of 2.8 Gpc (1 pc = 3.26 light years). Taken from [55].

These techniques are susceptible to human errors, leading to potential under-dedispersion or over-dedispersion. Such errors can obscure the true characteristics of the signal at emission. Additionally, the challenge is compounded by the uncertainty of the exact amount of dispersion correction needed, necessitating the exploration of a broad range of DM values. This requirement places a significant demand on computational resources.

Although autonomous dedispersion techniques exist, they still pose a significant challenge for large-scale, real-time data processing of FRBs, owing to their limited effectiveness and high computational demands. This issue will become increasingly critical with the commissioning of new FRB experiments such as the Square Kilometer Array (SKA).

1.1 Motivation

In this thesis, we attempt to address some of the challenges associated with dedispersing FRBs by using machine learning (ML) models. Such a model, capable of predicting the DM from the observed changes in signals over time and frequency, would significantly refine this process. Employed immediately after the RFI removal, the model enables a more precise selection of DM values, optimizing subsequent analysis steps and potentially accelerating the detection of FRBs from the extensive datasets gathered by telescopes. This independent assessment of DMs by the model is instrumental in confirming the potential FRB signals. When the model's DM estimates align with those derived from traditional analysis, it significantly bolsters our confidence in the signal's classification as a true FRB. This enhancement in the selection process not only streamlines the identification of genuine FRBs but also ensures that our final list of candidates is more accurate and reliable. However, leveraging its ability to analyze signal patterns – learned from extensive training on frequency-time images (i.e., the so-called "waterfalls") – the model provides an additional and significant advantage in the dedispersion process. The value of the ML model in identifying FRBs indeed extends well beyond the initial dedispersion phase.

After correcting candidate FRB signals to account for dispersion and converting them into a format that is easier to handle (e.g., a time series), the data are subjected to further processing. This includes manipulations like smoothing (i.e., filtering through down-sampling) to reduce noise, normalization to adjust signal strength, and matched filtering to better distinguish FRBs from other noise signals. These steps collectively refine the list of FRB candidates, setting the stage for a crucial reassessment.

Because the large distances to FRBs make it impossible to directly identify them and their environments using telescopes, the best one can do to discover signatures that provide clues about their nature is to study their "spectro-temporal" characteristics. Accordingly, researchers at Western have developed the so-called triggered-relativistic-dynamical-model (TRDM), which is very general in nature and applicable to a wide range of physical models aimed at explaining FRBs [46]. The TRDM was shown to recover known spectro-temporal characteristics such as the inverse scaling of a burst's duration with frequency and the downward drift in frequency of consecutive sub-bursts. Most importantly, and as we will see in Sec. 2.1.5, the TRDM predicted the existence of a new relation now known as the "sub-burst slope law," an inverse relationship between the sub-burst slope and duration in the FRB waterfall (the intensity as a function of frequency and time). This can be clearly visualized in the left panel of Fig. 1.2, where the narrower Burst #3 has the steepest slope (i.e., frequency/time). The most recent and wide-ranging verification of the sub-burst slope law over nine distinct FRB sources, where the frequency normalized sub-burst slope is plotted against the burst duration, is shown in the right panel of the figure [8]. The discovery of the sub-burst slope law and its emergence from a simple physical model is important because it imposes stringent constraints on any theory or model put forth to explain FRBs.

However, this type of analysis requires a precise dedispersion process, since many of the parameters recovered are adversely affected by an imprecise estimate of the DM. For example, an error in evaluating the DM will have a direct impact on the sub-burst slope (frequency/time) measured on FRBs such as those shown in Fig. 1.2. That is, too high a DM will increase the slope, and vice-versa. A lot of efforts were and are still put into developing robust de-dispersion techniques which, invariably, result in a lengthy and painstaking process that must be done on a "one-by-one" basis for each waterfall to ensure precise DM estimates. It is therefore highly desirable to train ML networks to automate the dedispersion to greatly speed up this process, as there is now a need to apply the TRDM analysis to the several thousands of available FRB signals from a growing number of sources. Through this application, the ML model emerges as a powerful tool, enriching the detection and analysis of FRBs with sophisticated data-driven insights.

By integrating ML into the FRB analysis, past the detection workflow, we move beyond traditional astrophysical approaches, embracing advanced data analysis techniques. This innovation not only facilitates the management of vast volumes of astronomical data but also

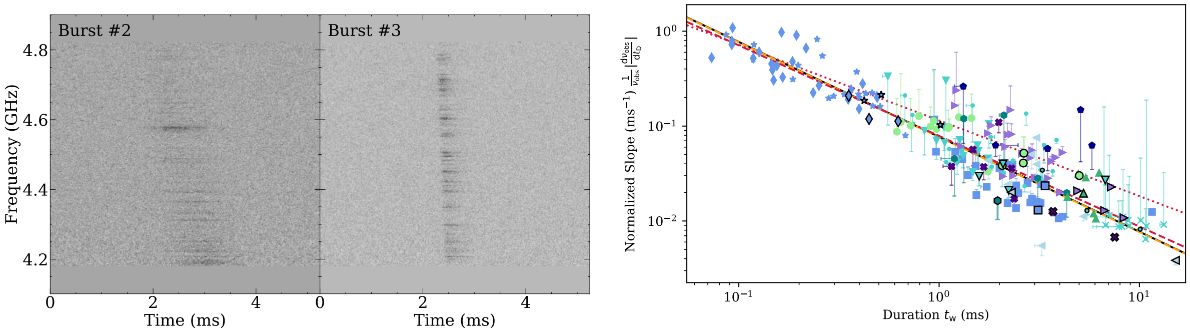

Figure 1.2: *Left:* Two dedispersed FRBs from the source FRB121102A; in these "waterfalls" the intensity is shown (grey scale) as a function of frequency and time. We can see that the narrower Burst #3 has the steepest slope (i.e., frequency/time). This behaviour was first theoretically predicted and verified in [46] and named the sub-burst slope law. *Right:* The most recent and wide-ranging verification of the sub-burst slope law over nine distinct FRB sources. The frequency normalized sub-burst slope is plotted against the burst duration, both covering more than two orders of magnitude in extent. The broken black/orange line is the theoretical prediction from the TRDM. Taken from [8].

improves our ability to identify and study these brief cosmic phenomena. The adoption of sophisticated models underscores a significant advancement, illustrating how the fusion of cutting-edge technology with space science is unlocking new realms of cosmic exploration.

1.2 Thesis Outline

The remaining of this thesis is structured as follows. In Chapter 2, we provide an in-depth background on FRBs and their characteristics, as well as discussing various ML algorithms and deep learning architectures utilized throughout this thesis. Chapter 3 delves into a summary of previously published works, covering research that applies ML to FRB data, offering a comprehensive understanding of this field's current state. The methodology used for our work is presented in Chapter 4, detailing the data generation process and various approaches employed for estimating DMs using different techniques. Chapter 5 covers the outcomes of our trained models and an analysis of their performance, as well as a discussion on and an assessment of our application of ML techniques to the problem of the dedispersion of FRBs. Finally, the thesis ends in Chapter 6 with a summary, our concluding thoughts, and suggestions for future directions for this line of research.

Chapter 2

Background

In this part of our study, we provide a more detailed look into the different components and parameters that characterize FRBs, and play crucial roles in their detection and analysis. Understanding these characteristics is essential for interpreting FRB data accurately and lays the foundation for the application of ML techniques.

Currently, the catalog of known FRB sources is limited, and the dataset is not large enough to effectively train an ML algorithm. This necessitates the simulation of FRBs with accurate properties to create a sufficiently large and representative dataset for training ML models for our intended goal (see below). Thus, it is crucial to simulate FRBs accurately, requiring a deep understanding of their intrinsic characteristics. Up to now, ML has been predominantly applied to classification tasks, such as distinguishing between different types of FRBs. This focus on classification provides valuable insights into how ML can be tailored for astrophysical data analysis. Examining these applications allows us to better understand the potential of various ML techniques, paving the way for their adaptation to our regression task: finding the optimal DM for FRBs.

In this section, we will go over the necessary background needed to simulate an FRB, setting the stage for a discussion on related works that showcase how previous research has utilized ML to address challenges in FRB analysis, and drawing connections to how these methods can be extended and applied to our specific objective of DM estimation.

2.1 Exploring FRB Characteristics

FRBs exhibit several defining characteristics that have intrigued astronomers since their discovery. These features, along with other external factors, provide clues to the FRBs' origins and the nature of the cosmos.

2.1.1 Dispersion Measure (DM)

As previously mentioned, the journey of FRB signals through space is influenced by their interaction with free electrons, leading to a phenomenon known as dispersion. Dispersion causes signals at lower frequencies to arrive later than those at higher frequencies, which is a key signature of FRBs. The DM quantifies the total column density of free electrons encountered along the signal's path (see equation 2.3 below) and is an important quantity for astronomers trying to pinpoint the FRB's origin. For instance, an FRB traveling from a distant galaxy will accumulate a higher DM value than one originating within the Milky Way, due to encountering a denser column of electrons.

The time delay Δt (in ms) between two frequencies ν_1 and ν_2 (in GHz) of an FRB due to dispersion is given by

$$\Delta t\,(\mathrm{ms}) = a\,\mathrm{DM}\left(\frac{1}{\nu_1^2} - \frac{1}{\nu_2^2}\right) \tag{2.1}$$

with

$$\begin{aligned} a &= \frac{e^2}{2\pi m_e c} \times 1\,\mathrm{pc} \\ &= 4.148\,806\,4239(11)\ \mathrm{GHz}^2\,\mathrm{cm}^3\,\mathrm{pc}^{-1}\,\mathrm{ms}, \end{aligned} \tag{2.2}$$

where e, m_e, c are the electron charge and mass, and the speed of light, respectively, while the DM is expressed in units $\mathrm{cm}^{-3}\,\mathrm{pc}$ in equation (2.1) [31] ($1\,\mathrm{pc} \simeq 3.26$ light years). The first relation highlights the inverse square relationship between frequency and time delay, which is precisely verified through observations of FRBs and pulsars. It is, for example, readily observed in the signal's curvature in Fig. 1.1, with $\Delta t > 0$ when $\nu_1 < \nu_2$ and vice-versa.

As previously mentioned, the definition of the DM is straightforward from the integration of the electron density n_e along the path of propagation of total length d

$$\mathrm{DM} = \int_0^d n_e(l)\,\mathrm{d}l, \tag{2.3}$$

thereby yielding a direct measure of the main component of the ionized medium's column density between the FRB source and Earth [44].

2.1.2 Scattering

Scattering is akin to the way light beams scatter when passing through a prism or a dense fog, causing signals to take indirect paths and arrive at slightly different times. This effect is particularly pronounced in FRBs due to their passage through the uneven interstellar medium. The scattering timescale τ_s can be expressed as:

$$\tau_s = \tau_0 \left(\frac{\nu_{\mathrm{ref}}}{\nu}\right)^n, \tag{2.4}$$

where $n = 4$ or 4.4 is usually adopted, while ν_{ref} is often set to 1 GHz and τ_0 is therefore the scattering time-scale at that frequency (usually on the order of a ms). This relation indicates that the scattering's impact on signal timing is strongly frequency dependent, offering insights into the density and turbulence of the medium through which the FRB has traveled. As was the case for the time delay due to the DM, we find that scattering is strongest at lower frequencies.

2.1.3 Scintillation

Scintillation describes the rapid fluctuations in signal intensity, similar to the twinkling of stars seen from Earth. This effect is more pronounced in FRBs due to their coherent, compact nature. For example, an FRB signal passing near a dense stellar object might exhibit enhanced scintillation due to the object's gravitational field affecting the signal path. The amplitude of scintillation A is modeled as

$$A = \cos\left[2\pi N_{\text{scint}}\left(\frac{\nu_{\text{ref}}}{\nu}\right)^2 + \phi_{\text{scint}}\right]. \tag{2.5}$$

Here, N_{scint} represents the number of scintillation events, and ϕ_{scint} introduces a phase shift, together simulating the dynamic nature of signal intensity fluctuations due to scintillation.

2.1.4 Pulse Width of an FRB

The observed pulse width of an FRB, denoted as W, is determined by a combination of factors that contribute to its broadening. These factors include the intrinsic properties of the signal itself, the characteristics of the observing instrument and the effects of the interstellar medium through which the signal propagates. To conceptualize this, one might consider the analogy of a flashlight beam passing through varying densities of fog; the denser the fog, the more dispersed or "spread out" the beam becomes. In the context of FRBs, this broadening of the signal can be attributed to several key elements detailed in the equation [44]

$$W = \sqrt{t_{\text{w}}^2 + t_{\text{samp}}^2 + \Delta t_{\text{DM}}^2 + \Delta t_{\text{DMerr}}^2 + \tau_s^2}, \tag{2.6}$$

where

- t_{w} stands for the intrinsic pulse width, which is the fundamental duration of the burst as emitted from its source. This width is the purest representation of the signal's initial form, unaffected by external factors.
- t_{samp} corresponds to the sampling time, which is defined by the temporal resolution of the data acquisition system. It reflects how frequently the observational data is collected and can artificially broaden the signal if the sampling rate is not sufficiently high.
- Δt_{DM} represents the dispersive delay variance across the measurement bandwidth due to the DM, illustrating how signals of different frequencies are delayed by varying amounts as they pass through the ionized medium.
- Δt_{DMerr} accounts for the error in correcting the dispersive delay, highlighting the limitations of de-dispersion techniques in perfectly reconstructing the original signal.
- τ_s is the scattering time scale, quantifying the broadening effect resulting from the scattering of the signal's components as they traverse the inhomogeneous interstellar medium.

Each term under the square root in this equation accounts for a distinct source of pulse broadening, collectively forming a comprehensive model that elucidates the observed width of an FRB pulse. Equation (2.6) underscores the multifaceted nature of FRB observations, where both intrinsic and extrinsic factors play pivotal roles in shaping the detected signal.

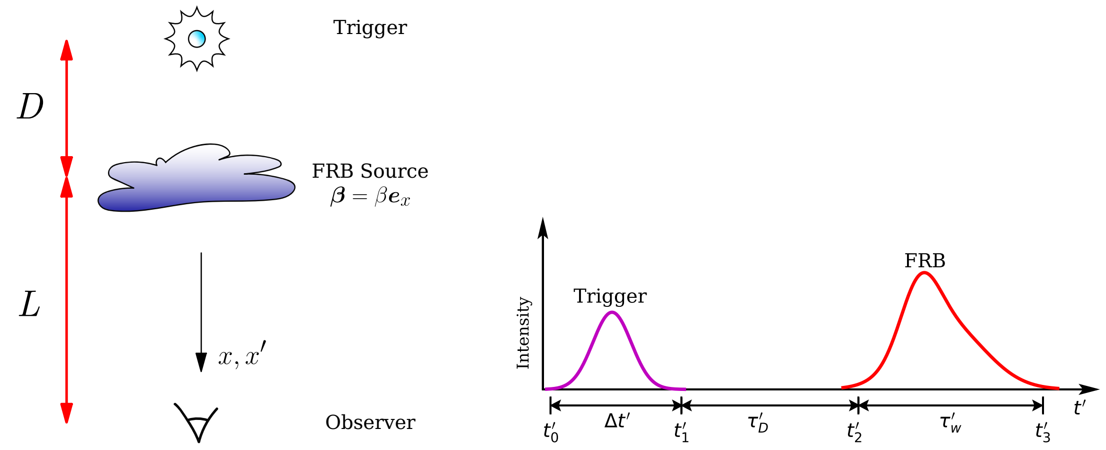

Figure 2.1: The framework of the TRDM of Rajabi et al. (2020) [46]. *Left:* The alignment of the trigger source, the FRB source and the observer with the FRB source is potentially moving at a relativistic speed $\mathbf{v} = \beta c\, \mathbf{e}_x$ relative to the observer. *Right:* Temporal sequence of the trigger and FRB signals in the rest-frame of the FRB source. The trigger stimulates the emission of a stronger FRB of duration τ'_w after a delay of τ'_D; amplitudes are not to scale. Taken from [46].

2.1.5 The sub-burst slope law

As mentioned earlier, the most important result stemming from the TRDM is the prediction, and subsequent verification, of the sub-burst slope law of repeating FRBs [46]. Within the framework of the TRDM the physical system producing an FRB is kept as simple and as general as possible, but has the following three ingredients: *i)* a trigger source that emits a short burst (e.g., a pulsar or a magnetar), *ii)* the FRB source, which is stimulated by the arrival of the short burst from the trigger, and *iii)* an observer who detects the FRB. Also, these three components are perfectly aligned while the FRB source is potentially moving at relativistic speed relative to the observer; see the left panel of Fig. 2.1. As shown in the right panel of the figure, the arrival of the trigger signal at the FRB source stimulates the emission of a stronger FRB of duration τ'_w after a delay of τ'_D, which is then detected by the observer at a later time.

Considering the relativistic transformations between the frames of the trigger and FRB sources, as well as the observer's, one arrives at the sub-burst slope law

$$\frac{1}{\nu_\mathrm{obs}} \frac{d\nu_\mathrm{obs}}{dt_\mathrm{D}} = -\frac{A}{t_\mathrm{w}}, \tag{2.7}$$

where $A \equiv \tau'_\mathrm{w}/\tau'_\mathrm{D}$ is considered to be approximately constant for a given FRB source, while ν_obs, t_w and t_D are respectively the FRB's frequency, duration and delay before arrival, all in the observer's frame. The sub-burst slope $d\nu_\mathrm{obs}/dt_\mathrm{D}$ is readily observed for the two dedispersed FRB in the left panel of Fig. 1.2. The width of the burst t_w (more generally W; see equation 2.6) is also a measurable quantity, which then leads to the mapping of equation (2.7), as shown in the right panel of Fig. 1.2 for nine different FRB sources.

These elaborations provide a more tangible understanding of the complex phenomena associated with FRBs. Each characteristic offers a unique perspective on the astrophysical conditions influencing FRBs, underscoring the importance of comprehensive analysis in unraveling the mysteries of these cosmic events.

2.2 Machine learning (ML)

Over the past few decades, ML has rapidly advanced as a sub-field of artificial intelligence. Primarily, it consists of developing algorithms that process input data gathered from past experiences to predict target values. The predictions could be continuous variables, for example, predicting real estate prices, or categorical values, for example, knowing whether a tumor is benign or malignant. It is thus quite versatile, with a lot of variances and approaches tailored to applications. There are four primary categories into which we can classify ML algorithms, which we briefly describe here.

Supervised Learning

Supervised learning involves using labeled data, where each data point is associated with the correct answer. The data is fed into a model, which uses optimization algorithms such as gradient descent to modify its parameters iteratively, to find the optimal weights that minimize the cost function J. A commonly used form of the cost function for regression tasks is the Mean Squared Error (MSE), expressed as

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left[h_\theta \left(x^i \right) - y^i \right]^2 . \tag{2.8}$$

In this formula, m denotes the number of training examples, x^i represents the input features of the i^{th} training example with y^i its the true label, $h_\theta\left(x^i\right)$ is the model's predicted value for the i^{th} input, and θ stands for the parameters of our model. This could alternatively be denoted as follows, in more general terms, where L is a loss function, $\hat{y}^i$ is the model predicted output and N is the number of training samples

$$J(\theta) = \min_\theta \frac{1}{N} \sum_{i=1}^{N} L(y^i, \hat{y}^i). \tag{2.9}$$

This is exactly equivalent to the definition provided by Tom Mitchell [38], "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E." This definition outlines the main components of learning, including the task, performance measure, and experience.

Once the model has been trained, it can be applied to new data. The model employs the patterns and relationships it learned during the training phase to predict the labels for these new data. The ability to generalize from the training data to new instances is the foundation of the predictive power of supervised learning.

Unsupervised Learning

Unsupervised learning is a type of ML that deals with unlabeled data, in contrast to supervised learning. Clustering and dimensionality reduction are the most commonly used algorithms in unsupervised learning. The model discovers patterns in the provided unlabeled data. Principal Component Analysis [1] and K-Means [33] are two popular algorithms in this family (see below).

Semi-Supervised Learning

Semi-supervised learning stands between supervised and unsupervised learning, using a combination of a small amount of labeled data and a large amount of unlabeled data. This approach helps to improve learning accuracy when acquiring labeled data is costly or time-consuming.

Reinforcement Learning

Reinforcement learning is an algorithm that learns through an iterative process of trials and feedback. Within a predefined space, with clear objectives and a range of actions it can take, the algorithm tries different strategies, evaluating the outcomes to identify the most effective ones. The approach is similar to learning through direct experience, where the algorithm adjusts its strategy based on the consequences of its actions, progressively improving its decision-making process to achieve the best result.

2.2.1 Tree-Based Models

Decision Trees are the main component of Tree-Based Models. A Decision Tree is a non-parametric supervised ML algorithm that mimics the structure of a tree, starting from a parent node commonly named the root node, and going down to the leaves; see Fig. 2.2. On each step, based on different measures such as Mutual Information or Gini Impurity, a feature would be placed in a node that expands the tree to sub-nodes [40]. Leaves are our interested values as the outcome of our algorithm. Despite their simplicity, decision trees are the foundation for more advanced techniques [50].

Random Forest

Random Forest is an ensemble learning algorithm that is used for a variety of tasks. The algorithm constructs several decision trees during the training process known as the decision tree ensemble method where each tree is constructed using a random subset of features and different bootstrap samples of the data; see Fig. 2.3 [42]. This technique combines base models to create a more accurate and robust predictive model. By aggregating the predictions of multiple decision trees, the ensemble method can mitigate overfitting and enhance the overall accuracy of the model [7].

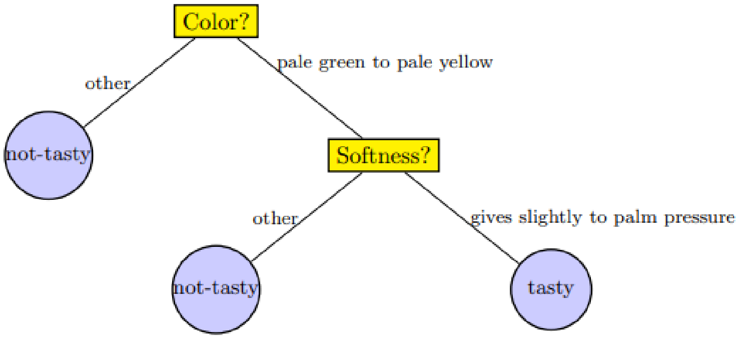

Figure 2.2: Illustration of a basic decision tree model to determine the tastiness of papayas [50].

AdaBoost

AdaBoost, or Adaptive Boosting, is another ensemble technique that combines multiple weak learners into a strong learner to improve performance and accuracy [20]. A weak learner is a model that is too simple to be used on its own. For example, decision trees with a single split, known as decision stumps, are a popular choice for weak learners in AdaBoost due to their effectiveness and simplicity. The process of training begins by assigning equal weights to all instances in the training dataset. AdaBoost then goes through several rounds of training weak learners, adjusting the weights of the instances that were misclassified by the previous model. Each weak learner is assigned a weight proportional to its accuracy, and these learners are combined into a weighted sum that constitutes the final model.

XGBoost

XGBoost (Extreme Gradient Boosting) is another ML algorithm well-known for its gradient-boosted framework. It uses advanced regularization techniques to enhance the model's generalization and combines multiple models' capabilities into a single robust model using an ensemble learning approach [11]. In regression tasks, XGBoost commonly utilizes a squared error loss function or mean squared error as its objective. The algorithm improves the accuracy by adding decision trees to the ensemble and fine-tuning each tree to minimize the cumulative loss.

LightGBM

LightGBM is another member of the gradient-boosting family. It stands out because of its unique approach to processing large datasets efficiently. Key characteristics of LightGBM are Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB). GOSS allows LightGBM to focus on the most informative data points, essentially prioritizing harder-to-predict instances, which enhances learning efficiency. On the other hand, EFB reduces

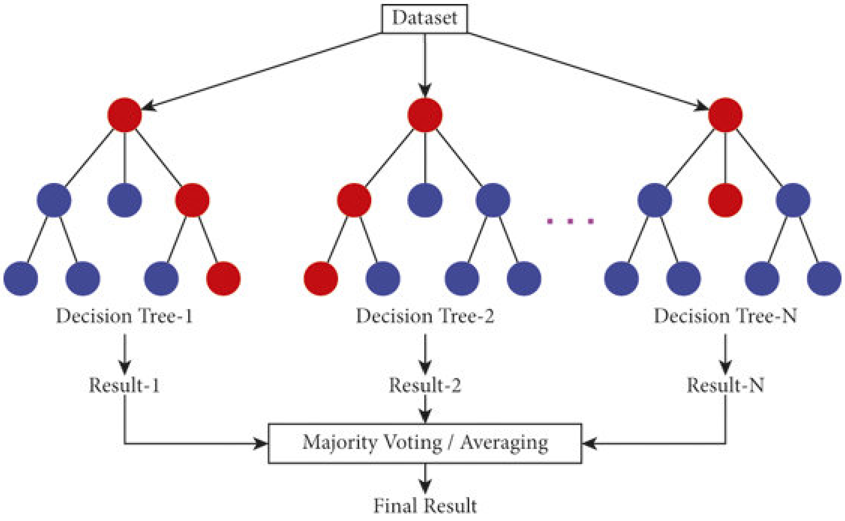

Figure 2.3: Illustration of a random forest model [29].

the feature space of high-dimensional data by grouping features that are mutually exclusive, resulting in faster training [28].

2.2.2 Support Vector Machines

Support Vector Machine (SVM) is a supervised ML algorithm [41]. As was previously discussed about supervised learning, such algorithms use labeled data to learn mapping input features to output targets by fitting a hyperplane that maximizes the margin between different categories in the case of a classification task. In an n-dimensional space, a hyperplane can be described as a flat surface that extends infinitely in all directions and has one less dimension than the space itself, making it an $(n-1)$-dimensional entity. This hyperplane provides a boundary that can divide the space into two distinct parts. In the context of SVM, the hyperplane is defined by the equation $w^\top x + b = 0$, where:

- w represents the weight vector perpendicular to the hyperplane.
- x denotes the input feature vectors.
- b is the bias term.

The primal form of the SVM objective function for a training dataset $S = \{(x_i, y_i); i = 1, \ldots, m\}$ that is linearly separable is formulated as:

$$\begin{aligned} &\underset{w,b}{\text{minimize}} && \frac{1}{2}||w||^2 \\ &\text{subject to} && y_i\left(w^\top x_i + b\right) \geq 1, \quad \forall i = 1, \ldots, m. \end{aligned}$$

Where $||w||$ is the Euclidean norm of vector w. The role of y_i, which takes values from $\{-1, +1\}$, is to indicate the class of each input vector x_i, thereby guiding the placement of the hyperplane to ensure it separates the classes with a maximum margin [39]. When S is not linearly separable, the method involves either projecting the input features into a higher-dimensional space to achieve linear separability or maintaining the original feature space. In such cases, a slack variable ξ is introduced to allow some degree of misclassification [57]. This leads to a modified optimization formulation:

$$
\begin{aligned}
&\min_{w,b,\xi} \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i \\
&\text{subject to} \\
&y_i\left(w^\top x_i + b\right) \geq 1 - \xi_i, \\
&\xi_i \geq 0, \quad \forall i = 1, \ldots, m,
\end{aligned}
$$

where C is a regularization parameter employed to balance the margin maximization with the penalty for the misclassifications.

2.2.3 Clustering Algorithms

In the realm of ML, clustering algorithms play a pivotal role in unsupervised learning tasks. Their primary objective is to group data points based on their similarities, without prior labeling or categorization. This approach enables the discovery of inherent structures or patterns within the data.

K-means

K-means is one of the simplest and most widely used clustering algorithms. It partitions the dataset into K distinct clusters based on the distances between data points and the centroids of the clusters. The algorithm iteratively updates the centroids until it minimizes the within-cluster variance, often measured as the Euclidean distance between data points and their corresponding cluster centroids [33].

HDBSCAN

HDBSCAN is an extension of DBSCAN, which is an algorithm that clusters data based on density. HDBSCAN converts DBSCAN into a hierarchical clustering algorithm. It then uses a technique to extract flat clusters based on the stability of clusters over the hierarchy [9].

2.2.4 Dimensionality Reduction Techniques

Dimensionality reduction plays a critical role in ML by simplifying high-dimensional datasets, enhancing visualization, and sometimes improving algorithm performance by removing irrelevant features or noise. This section elaborates on three principal techniques: PCA, t-SNE, and UMAP.

Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a linear technique for reducing the dimensionality of a dataset while maximizing variance. By identifying the principal components with the highest variance, PCA summarizes the data with fewer features. This technique is particularly effective for datasets where linear relationships are predominant [27].

t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a non-linear, probabilistic technique primarily used for the visualization of high-dimensional data in two or three dimensions. By converting high-dimensional Euclidean distances into conditional probabilities that represent similarities, t-SNE maintains local data structures and can reveal complex structures in the data [56].

Uniform Manifold Approximation and Projection (UMAP)

Uniform Manifold Approximation and Projection (UMAP) is a relatively recent technique that can be used for dimensionality reduction and as a non-linear visualization tool. UMAP maintains the global structure of the data more effectively than t-SNE, providing a comprehensive view that balances both local and global data relationships. It is best suited for datasets where both local and global structures are important. Due to its efficiency and scalability, UMAP is advantageous for large datasets [37].

2.3 Artificial Neural Networks

Inspired by the brain's architecture, which is composed of neurons communicating through signals, the first mathematical model of this kind was proposed to mimic biological neurons. This foundational model, capable of producing a binary output from binary inputs, introduced the concept of computational units [36]. Building upon this, the Perceptron [47], the very first artificial neural network, was introduced. It consisted of a single layer made up of multiple threshold logic units (TLU). A TLU acts as a neuron that receives numerical inputs, each input connection having an assigned weight, and it produces an output based on a linear combination of these inputs and weights, which is then passed through an activation function. In the Perceptron, every TLU is connected to each input, forming a fully connected layer. The input layer comprises the inputs, while the layer of TLUs generates the final outputs, thus referred to as the output layer. The output of a fully connected layer can be mathematically represented as:

$$\mathbf{y} = \phi\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right) \tag{2.10}$$

where:

- $\mathbf{y}$ is the output vector of the layer.
- ϕ denotes the activation function applied element-wise.
- $\mathbf{W}$ is the weight matrix of the layer.

- **x** is the input vector to the layer.
- **b** is the bias vector.

The perceptron training algorithm enhances neural connections by modifying connection weights to lower prediction errors. This approach processes each data point one by one, making predictions and then changing the weights of neurons that got predictions wrong. This helps the model get better at making correct predictions. The changes are made according to this rule [23]:

$$w_{i,j}^{(\text{new})} = w_{i,j} + \eta(y_j - \hat{y}_j)x_i \tag{2.11}$$

where:

- $w_{i,j}$ is the weight from the i^{th} input to the j^{th} neuron.
- x_i is the i^{th} input value for the current data point.
- $\hat{y}_j$ is the predicted output of the j^{th} neuron for the current data point.
- y_j is the actual target output for the j^{th} neuron.
- η is the learning rate, controlling the update magnitude.

This rule guides the algorithm to reduce mistakes by repeatedly tweaking weights according to the difference between what it predicts and the real results.

2.3.1 Multi Layer Perceptron

The primary limitation of a single perceptron is its inability to handle data that is not linearly separable. To address this, multiple perceptrons are stacked together to form a multi-layer perceptron (MLP) architecture, where each layer is connected to the following layer in the network. This transition from single-layer perceptrons to multi-layer perceptron architecture enables models to learn non-linear and complex patterns in data. However, this additional capability requires a more sophisticated training approach [23].

For training MLPs, the network weights and biases are initialized randomly. However, it is usually better to initialize these parameters using techniques such as Xavier [22] or He initialization [24] for faster convergence and addressing gradient vanishing or exploding problems. Following weights initialization, each input data is fed to the model, and the output of each neuron is calculated. Calculating the output involves taking the weighted sum of the neuron inputs and passing it through an activation function. The process of passing data through a neural network in a forward direction is called a feedforward pass.

The predicted output of the model resulting from this feedforward operation is then compared with the real output, and used to calculate the model's prediction error with a loss function (or cost function), such as the mean squared error for a regression model, or the cross-entropy loss for a classification model. The training objective then is to minimize this error, and one approach is to use the Backpropagation algorithm. Backpropagation is an algorithm used to minimize the error of the network by calculating the gradient of the cost function with respect to each parameter in the network. This tells us how the error will change as each weight in each neural network layer is adjusted [48].

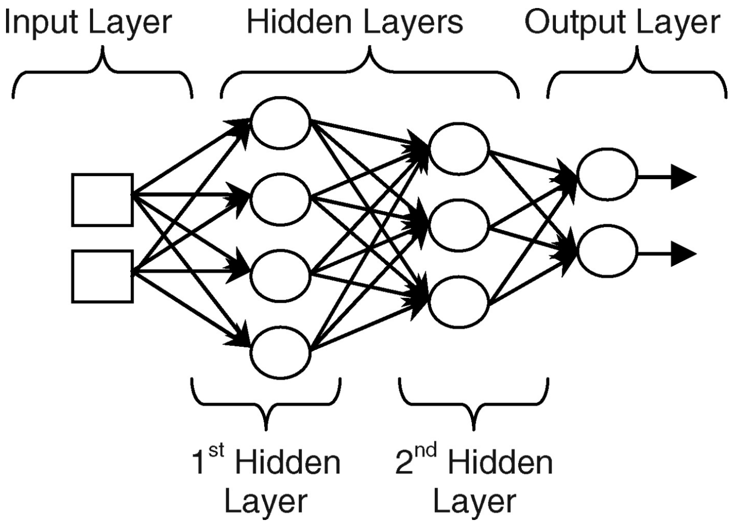

Figure 2.4: Illustration of an MLP with two hidden layers and two outputs [30].

2.3.2 Activation Functions

Activation functions are a big part of the field of neural networks, as they help solve non-linear problems. They get an input, the weighted sum from the previous layer, and compute a non-linear output. Without non-linear activation functions, neural networks would be not much different from basic single-layer perceptrons, with their problem-solving capabilities greatly diminished [23]. While we could just use an activation function like the Identity function, which just returns the input as is, it is most commonly used in the output layer in tasks like regression. In this section, we provide a brief overview of activation functions commonly used in modern neural network architectures.

Sigmoid

This function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.12}$$

maps any input into a range between 0 and 1. This makes the Sigmoid function useful for models where the output is interpreted as a probability. Despite it being commonly used in binary classification problems and the output layers of neural networks, the Sigmoid function can lead to vanishing gradients because it saturates for largely positive or negative values. This leads to a point that updates to the weights become insignificantly small, potentially slowing down or stopping the training process.

Softmax

The Softmax activation function plays a crucial role in the field of neural networks, especially in multi-class classification problems. It is defined by the formula

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{2.13}$$

where z_i represents the input to the function for class i, and the denominator is the sum of the exponential values of all inputs. This function converts logits, or raw predictions from a neural network, into probabilities by taking the exponential of each input value and normalizing these values by dividing by the sum of all the exponentials. The result is a probability distribution over all possible classes, with each value ranging between 0 and 1, and the sum of all probabilities being 1. The Softmax function is typically used in the output layer of a neural network model for multi-class classification, where it provides a clear and interpretable output as probabilities for each class [6].

Rectified Linear Unit (ReLU)

Rectified Linear Unit (ReLU) is one of the most widely used activation functions in deep neural networks. Essentially, ReLU functions output the input directly if it is positive, and outputs zero if it is negative. This allows it to alleviate the vanishing gradient problem (common in deep networks), since its gradient is either zero (for negative inputs) or one (for positive inputs). It also allows the network to model complex patterns using non-linearities while imposing no extra computational burden.

2.3.3 Optimizers

The choice of an optimization algorithm is crucial for determining the efficiency with which the model learns by finding the optimal values for its parameters (weights) that minimize the error in mapping inputs to outputs. The choice of an optimizer not only influences the speed of convergence and the accuracy of the model but also influences whether the model converges at all. Various optimizers with distinct algorithms have been developed to address these challenges. These include Gradient Descent (GD), Stochastic Gradient Descent (SGD), SGD with momentum, Mini-Batch Gradient Descent, RMSProp (Root Mean Square Propagation), AdaDelta, Adagrad (Adaptive Gradient Algorithm), Adam (Adaptive Moment Estimation), and more sophisticated methods like Conjugate Gradient, BFGS (Broyden–Fletcher–Goldfarb–Shanno algorithm), and L-BFGS (Limited-memory BFGS). In this section, we discuss the theoretical and practical nuances of a selection of these optimizers that are predominantly utilized in the field.

Gradient Descent

The procedure of repeatedly evaluating the gradient and then performing a parameter update is called Gradient Descent (GD). By calculating the gradient of the loss function with respect to the parameters, GD iteratively adjusts the parameters in the direction that minimally decreases the loss and provides the steepest direction to reach the local minimum of the loss function. It starts with some coefficients, and searches for cost value smaller than what it is now. It moves towards the lower weight and updates the value of the coefficients. The process repeats until the local minimum is reached. A local minimum is a point beyond which it cannot proceed.

While GD works well for small datasets, its efficiency significantly diminishes for large datasets due to the necessity of computing gradients across the entire dataset at each iteration.

Moreover, GD works well for convex functions, however, it struggles with non-convex functions as it cannot guarantee global minimum discovery and may get stuck in local minima. GD is guaranteed to converge to the global minimum for convex loss surfaces and to a local minimum for non-convex surfaces. GD is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. The update rule for the parameters in the vanilla version of GD is given by:

$$\theta_{\text{new}} = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{2.14}$$

where θ represents the parameters of the model, η is the learning rate, $\nabla_\theta J(\theta)$ is the gradient of the loss function J with respect to the parameters θ.

Stochastic Gradient Descent (SGD)

This algorithm enhances the scalability of traditional Gradient Descent (GD) by updating the model's parameters using only a single example from the dataset at each iteration. This approach allows SGD to begin optimizing the cost function immediately with each example, making it significantly faster and more suitable for large datasets. Unlike GD, which requires the entire dataset to compute the gradient, SGD's incremental updates significantly reduce the computational burden.

However, these updates are based on a single example at a time, which introduces more noise into the optimization process compared to GD. This noise can cause the convergence path of SGD to be more erratic, often oscillating around the minimum rather than smoothly converging towards it. As a result, while SGD is generally faster than GD, especially with large datasets, its path to convergence can be less stable. The model may not settle precisely at the minimum but in its vicinity, reflecting the trade-off between speed and the precision of convergence.

The update rule for SGD can be represented as follows:

$$\theta_{\text{new}} = \theta - \eta \cdot \nabla_\theta J\left(\theta; x^{(i)}, y^{(i)}\right) \tag{2.15}$$

where θ represents the parameters of the model, η is the learning rate, $\nabla_\theta J\left(\theta; x^{(i)}, y^{(i)}\right)$ is the gradient of the loss function J with respect to the parameters θ, evaluated at a single example $\left(x^{(i)}, y^{(i)}\right)$.

Stochastic Gradient descent with momentum

Momentum is an enhancement to Stochastic Gradient Descent (SGD) that accelerates convergence by incorporating a fraction of the previous update vector into the current update. While Momentum uses more memory for a given batch size than SGD, it requires less memory than the RMSprop and Adam algorithms. This method significantly reduces oscillation and speeds up convergence, particularly in contexts where the loss function's surface curvature varies sharply. The momentum term is beneficial for navigating through flat regions and dampening oscillations in steep directions. However, adding a fraction of the previous update to the current one not only helps in faster convergence of the loss function by mitigating the slower computation times and the noisy path of SGD, which results in a higher number of iterations to reach the local minima, but it also necessitates careful management of the learning rate and momentum hyperparameters. Improper settings can lead to overshooting the minimum or instability in the convergence process. The learning rate should be decreased when a high momentum term is used because as the momentum increases, so does the likelihood of surpassing the optimal point, which might result in poor accuracy and even more oscillations. The update rule for this algorithm is:

$$\begin{aligned} v_t &= \alpha v_{t-1} + \eta \nabla_\theta J(\theta) \\ \theta_{\text{new}} &= \theta - v_t, \end{aligned} \tag{2.16}$$

where v_t is the velocity at time t, α is the momentum coefficient, η is the learning rate, $\nabla_\theta J(\theta)$ is the gradient of the loss function J with respect to the parameters θ.

Mini-Batch Gradient Descent

This algorithm optimally balances the robustness of Gradient Descent (GD) with the efficiency of Stochastic Gradient Descent (SGD). By updating parameters using a small, randomly selected subset of the training data, termed a mini-batch, at each iteration, this method significantly reduces the computational burden compared to GD, which uses the full dataset for each update. Compared to SGD, which updates parameters using a single example at a time, Mini-batch GD offers a more stable path to convergence by smoothing out the variance in parameter updates. This approach strikes a good balance between speed and accuracy: the updates to the cost function are noisier than with GD but smoother compared to the updates in SGD. This balance allows Mini-batch GD to efficiently navigate the cost landscape, achieving faster convergence than GD while maintaining a more consistent trajectory towards the minimum than SGD. However, it introduces the additional hyperparameter of mini-batch size, which requires careful tuning to achieve optimal performance. Generally, a mini-batch size of 32 is considered a good starting point, but the optimal size can vary depending on the specific problem and dataset. This method's main advantage lies in its ability to provide a good balance between computational efficiency and convergence stability, making it a popular choice for training neural networks and other ML models. The update rule for this algorithm is:

$$\theta_{\text{new}} = \theta - \eta \cdot \nabla_\theta J(\theta; X_{\text{mini}}, Y_{\text{mini}}), \tag{2.17}$$

where θ represents the parameters of the model, η is the learning rate, $\nabla_\theta J(\theta; X_{\text{mini}}, Y_{\text{mini}})$ is the gradient of the loss function J with respect to the parameters θ, evaluated on a mini-batch $(X_{\text{mini}}, Y_{\text{mini}})$.

Adam (adaptive moment estimation)

Adam is a further extension of stochastic gradient descent to update network weights during training. Unlike in SGD, which maintains a single learning rate for all updates, Adam updates the learning rate for each network weight individually. It combines the advantages of two other extensions of SGD, namely AdaGrad and RMSProp, by adapting the learning rate based on the first and second moments of the gradients. The first moment represents the mean, and the second moment represents the uncentered variance, meaning that the (square of the) mean is not subtracted. This approach not only automates the adjustment of the learning rate throughout training but also ensures a balance between fast convergence and stability, making it particularly beneficial in settings with large datasets and/or high-dimensional parameter spaces.

Adam differentiates itself by focusing on faster computation times, in contrast to SGD, which, despite potentially offering better generalization through its focus on individual data points, does so at the cost of slower computation speeds. Adam's formula incorporates decay rates for the averages of the gradients (denoted as β_1 and β_2), which facilitate adaptive step-sizes, effectively performing a form of learning rate annealing. Known for its popularity and efficacy, Adam is often the default optimizer in ML frameworks, attributed to its ability to handle sparse gradients and different scales of parameters efficiently.

However, it is noteworthy that Adam requires more memory for a given batch size compared to other optimizers, due to maintaining separate learning rates for each parameter through the storage of the first and second moments of the gradients. Despite this, its comprehensive approach introduces several hyperparameters, such as the learning rate and exponential decay rates for the moment estimates, which, while requiring tuning for optimal results, often perform adequately well with default settings. This makes Adam a user-friendly and effective choice for a wide range of deep learning applications. The formula for Adam optimizer is as follows:

$$\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1-\beta_1)\,\nabla_\theta J(\theta) \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2)\,[\nabla_\theta J(\theta)]^2 \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\
\hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\
\theta_{\text{new}} &= \theta - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t,
\end{aligned} \tag{2.18}$$

where m_t is the first moment vector (mean) which represents the average of the gradients, v_t is the second moment vector (uncentered variance) which represents the average of the squared gradients, β_1 and β_2 are the exponential decay rates for the moment estimates, typically set close to 1 (e.g., 0.9 and 0.999), η is the learning rate, ϵ is a small scalar (e.g., 10^{-8}) to prevent division by zero, $\nabla_\theta J(\theta)$ is the gradient of the loss function J with respect to the parameters θ, $\hat{m}_t$ and $\hat{v}_t$ are bias-corrected versions of m_t and v_t, ensuring they are unbiased at initialization. β_1^t and β_2^t are biased corrected versions of β_1 and $beta_2$.

2.4 Convolutional Neural Networks

MLPs are powerful algorithms. However, when the input size gets very large as in the case of images and videos, they may perform poorly due to the high number of required parameters for training and slower convergence. This challenge has led to the transition from MLPs to Convolutional Neural Networks (CNN), showing proven and effective results in handling such tasks. CNN are deep learning algorithms well-suited for Computer Vision tasks because of their innovative building blocks, inspired by how the human visual cortex processes data.

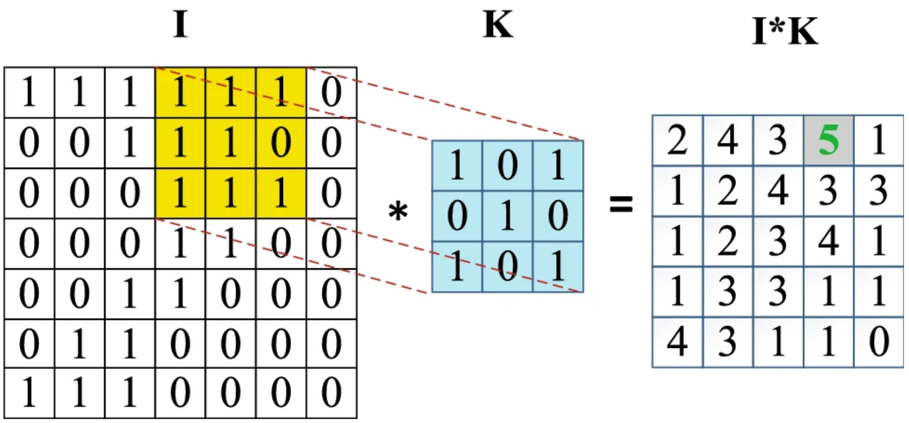

Figure 2.5: Illustration of a simple two-dimensional convolution Operation [43].

Focusing on pattern recognition and spatial hierarchies, their architecture is built around three key components: convolutional layers, pooling layers, and fully connected layers [32].

2.4.1 Convolutional Layer

Convolutional layers are necessary for processing visual information. Using this layer, the input image is scanned over a set of filters or kernels. Each filter can detect a specific feature by performing element-wise multiplication of the filter elements and the segment of the input that the kernel covers, known as the receptive field. The result of this operation is a feature map that represents the presence of the detected feature across different locations of the input. The two-dimensional convolution operation in CNN could be formulated as:

$$S\left(i, j\right) = \left(K * I\right)\left(i, j\right) = \sum_{m} \sum_{n} I\left(i - m, j - n\right) K\left(m, n\right), \tag{2.19}$$

where:

- $S\left(i, j\right)$: The output of the convolution at position $\left(i, j\right)$.
- K: The horizontally and vertically flipped kernel applied during the convolution.
- I: The input image or matrix to which the convolution is applied.
- m, n: Indices used to navigate through the kernel K.
- i, j: Indices used to navigate through the input image I.

This operation is performed for every element in the output matrix, effectively capturing local patterns within the input image. However, in most ML libraries convolution operation is implemented using the cross-correlation formula:

$$S\left(i, j\right) = \left(K * I\right)\left(i, j\right) = \sum_{m} \sum_{n} I\left(i + m, j + n\right) K\left(m, n\right). \tag{2.20}$$

The formula for cross-correlation is more straightforward to implement and easier to grasp while achieving the same result. The key differences are that it does not involve flipping the kernel and it changes the indexing approach for the image [23].

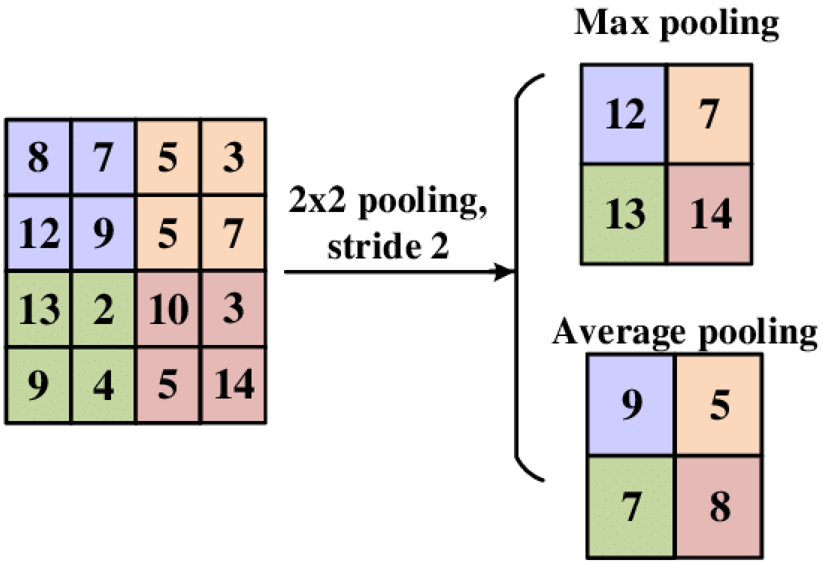

Figure 2.6: Illustration of max pooling and average pooling with their respective results [60].

2.4.2 Pooling Layer

One of the reasons that CNN is computationally efficient for computer vision tasks is due to the use of the pooling layers. These layers reduce spatial dimensions of feature maps, which in turn lowers computational costs while helping prevent overfitting by decreasing the number of model parameters. On top of that, networks are more robust against minor changes in input images by only focusing on the patterns. There are two commonly used pooling types: max pooling and average pooling. Max pooling selects the maximum value from the receptive field, keeping only the most prominent feature. In contrast, average pooling calculates the mean value within the window, maintaining a generalized representation of features. These two categories are shown in Figure 2.6. Stride is a user-defined hyperparameter that determines the movement step size of the pooling layer across the input image.

Building on the architectural fundamentals of CNNs, we go over the data flow within a CNN as depicted in Fig. 2.7. By understanding this pathway, we can appreciate how CNNs can capture and interpret the complex patterns found in visual data for tasks such as image recognition and classification. Initially, as each data point is passed to the network, convolutional layers apply filters to the input image to detect features (e.g., edges, textures). These filters are initially randomly initialized and then gradually adjusted during the training process to capture more complex and relevant patterns within the data. A pooling layer follows each convolutional layer, and the feature maps are passed through the pooling layers, reducing the spatial dimensions of the feature maps.

As the data progresses, it may go through additional convolutional and pooling layers, each time abstracting higher-level features. Eventually, the processed data (feature maps) are flattened and reach the fully connected layers. The fully connected layers, using these high-level features, predict an output. Afterward, the training process is similar to the training of MLPs, where the loss is calculated, and the weights are adjusted according to gradients using

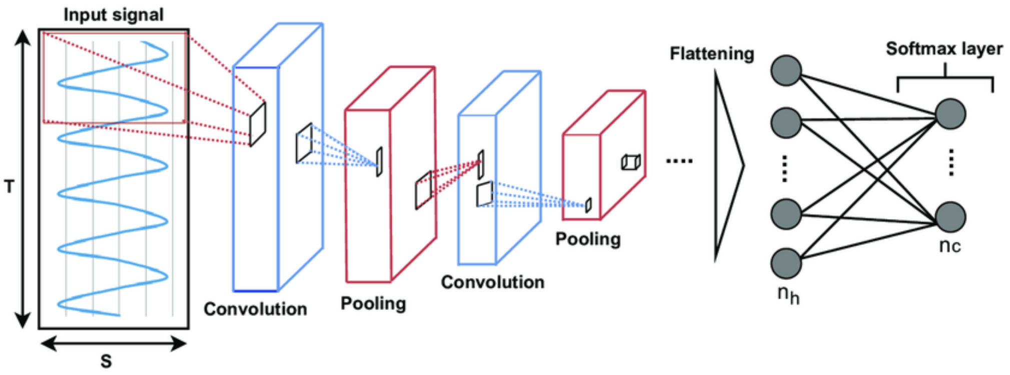

Figure 2.7: Illustration of data fellow through various layers—convolutional, pooling, and fully connected—to produce final results in CNNs [26].

an optimizer. This process is repeated for each batch of data until the loss is minimized. Now, this trained model could be used for different tasks; it will work seamlessly to predict new outputs because it has learned patterns in the data. However, there are some implementation details related to network generalization that we will discuss in the next section.

2.4.3 Dropout

Dropout is a regularization technique used to prevent overfitting in neural networks, particularly beneficial for large networks or training sessions that extend over long periods. These conditions typically increase the risk of the model fitting too closely to the training data, thus failing to generalize well to unseen data. By randomly omitting a subset of neurons along with their connections in every training iteration, dropout introduces variability in the network architecture. This variability forces the network to learn more robust features that are not dependent on a specific set of neurons, essentially simulating a form of model averaging or ensemble learning. Ensemble methods are known for their superior performance over single models, as they capture a broader range of data patterns by combining the predictions from multiple models. Similarly, dropout enhances model performance by emulating the training of numerous neural networks with different architectures.

The core principle behind dropout, as depicted in Fig. 2.8, involves randomly setting a portion of activations in the network to zero during the training phase. Typically, 50% of activations in a layer are dropped, though this rate is adjustable based on the specific needs of the network and is considered a hyperparameter of the dropout function. This process of selective activation nullification compels the network to distribute its learning over a wider range of neurons, thereby reducing the model's reliance on any individual neuron. It's akin to training the network to achieve redundancy and ensure that informative signals are captured by multiple pathways within the network architecture. Despite its effectiveness in enhancing generalization, one notable downside of using dropout is the substantial increase in training time. The introduction of randomness through dropout means that each iteration trains a somewhat different network configuration, leading to noisier updates and, consequently, a longer convergence time. This is because the network effectively explores a larger architecture space during training.

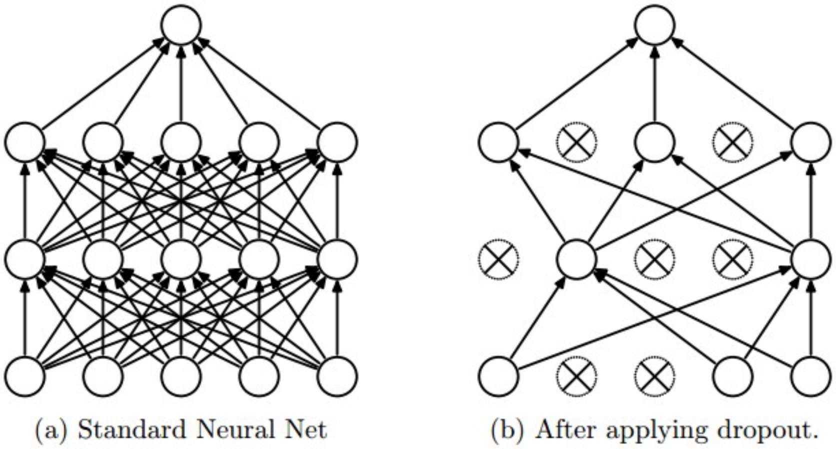

Figure 2.8: Illustration from the original Dropout publication [54] showcasing the concept. In the training phase, Dropout is akin to randomly selecting a subset of the network from the broader network architecture, updating only this subset's parameters in response to the input data. In the testing phase, Dropout is not utilized, embodying the process as an ensemble average prediction over the extensive collective of possible sub-networks.

To accommodate the changes introduced by dropout during training, adjustments are made during the test and inference phases to maintain consistency in the network's behavior. One approach is to scale the activations by their retention probability, ensuring that the expected sum of activations remains the same as it would without dropout. This method, known as "inverted dropout," scales activations during training, which simplifies the transition to test time by eliminating the need for additional scaling. At test time, instead of randomly omitting neurons, the activations are adjusted to reflect the average outcome of the dropout process. If the dropout probability is set to 0.5, for example, the activations are multiplied by 0.5 to compensate for the halved number of active neurons on average during training.

2.4.4 Batch Normalization

Normalization is a technique for speeding up and stabilizing the training of deep neural networks. It is considered a best practice to always employ normalization when training deep neural networks due to its numerous benefits. By normalizing each feature to have zero mean and unit variance, all features contribute equally to the training process. This is crucial because some features may have higher numerical values than others, leading to a biased network that favours features with larger values. Normalization, applied after the activation function step, addresses this issue by maintaining an unbiased environment for all features.

Another key advantage of normalization is its ability to reduce Internal Covariate Shift. Internal Covariate Shift refers to the change in the distribution of network activations that occurs as the network parameters are updated during training. By mitigating this shift, normalization significantly improves the training process. Additionally, techniques like Batch Normalization smooth out the loss surface, tightly bounding the magnitude of gradients and thereby enhancing the stability of the training process.

The impact of normalization extends to the efficiency of the training as well. It considerably decreases training time by facilitating faster optimization. This is achieved because normalization prevents the weights from becoming too large or exploding, keeping them within a specific range. Furthermore, although not its primary intention, normalization slightly contributes to the regularization of the network, providing a modest boost in preventing overfitting. Overall, the implementation of normalization in deep neural network training offers a combination of improved speed, stability, and performance, making it an indispensable technique in the field of deep learning.

Batch normalization is a technique to normalize activations across a mini-batch, stabilizing and expediting the learning process in neural networks. During training, it computes the mean and variance for each feature, adjusting the features to have zero mean and unit variance. The normalization step is integrated into the backpropagation algorithm. Learnable parameters gamma γ and beta β are introduced to allow the model to scale and shift the normalized output and enhance performance. The steps for applying batch normalization are as follows:

To centre the data, the mean activation across the mini-batch is computed using:

$$\mu_B = \frac{1}{m} \sum_{i=1}^{m} x_i. \tag{2.21}$$

The variance of the activations is calculated to understand the spread of the data as follows:

$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^{m} \left(x_i - \mu_B\right)^2 . \tag{2.22}$$

Each feature is normalized to have zero mean and unit variance. The normalized feature is computed using:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}. \tag{2.23}$$

Finally, to allow the network to learn an identity transformation if it is beneficial, two learnable parameters, γ (scale) and β (shift), are used to scale and shift the normalized feature, using the equation:

$$y_i = \gamma \hat{x}_i + \beta. \tag{2.24}$$

In these formulas:

- x_i is the input value for feature i in the mini-batch
- m is the number of examples in the mini-batch
- μ_B is the mini-batch mean
- σ_B^2 is the mini-batch variance
- $\hat{x}_i$ is the normalized input

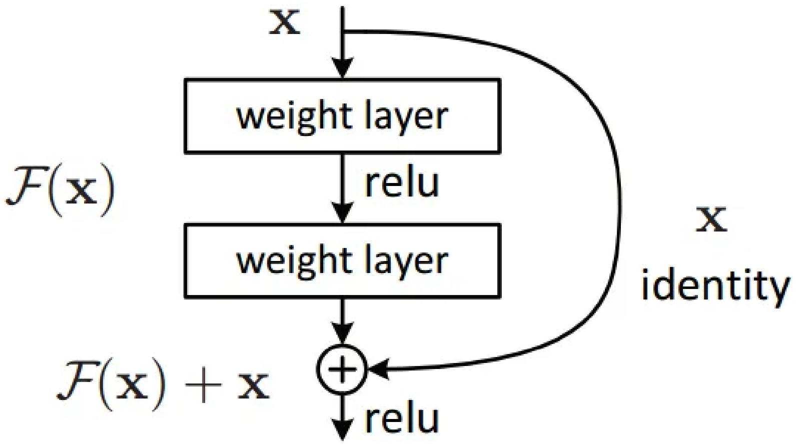
x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2.9: Illustration of a residual block from the original paper [25].

- y_i is the final output of the batch normalization layer
- γ and β are parameters learned during training that scale and shift the normalized value
- ϵ is a small constant for numerical stability.

During test time, the model utilizes global statistics rather than mini-batch-specific values to maintain consistency in predictions. The approach faces challenges with variable batch sizes, where a batch size of one leads to a variance of zero, thereby affecting normalization. Similarly, in distributed training, inconsistent batch sizes across different machines can result in varying γ and β values, compromising model performance.

In Recurrent Neural Networks (RNNs), the dynamic nature of recurrent activations requires potentially fitting a separate batch normalization layer for each time-step, which complicates the model and increases memory demands. This is due to each time-step's recurrent activations having distinct statistics, necessitating storage of these statistics throughout training.

Alternatives to batch normalization such as weight normalization, layer normalization, instance normalization, and group normalization, address some of these challenges, offering benefits in terms of training stability and performance across varying architectures and contexts.

2.5 Residual Neural Network

Residual Networks, or ResNets, represent a significant advancement in the design of deep neural networks. Introduced by He et al. in their 2016 paper [25], ResNets effectively address the challenges associated with training very deep networks. The key feature of ResNet is the introduction of 'skip connections' that allow inputs to skip over some layers and be added directly to the outputs of deeper layers. This is illustrated in Fig. 2.9, which shows a residual block.

A residual block employs the function $F(x) + x$, where x is the input and $F(x)$ represents the changes the network needs to learn, known as residual mapping. Adding x directly into the output using a skip connection is the key idea. This method preserves the input throughout the network's layers without changing it. The network then can focus on learning only the necessary adjustments, rather than reconstructing the input from scratch. One major benefit of the $F(x) + x$ setup is the enhancement of gradient flow. Skip connections help to directly pass gradients through several layers, mitigating the issue of vanishing gradients in deep networks. This strategy not only aids in stabilizing and speeding up the convergence during training but also improves the model's ability to generalize to new data. Furthermore, it allows the deeper layers to utilize features processed by earlier layers without the risk of feature degradation.

The original implementation of ResNets by He et al. introduced several variants, each with a different number of layers, to accommodate various depths needed for different tasks. The most common variants include ResNet-18, ResNet-34, ResNet-50, ResNet-101 and ResNet-152. The numbers denote the layers in each model, with ResNet-18 being the shallowest and ResNet-152 being the deepest.

These different variants have their layers organized into blocks, each containing multiple convolutional layers. The simpler models like ResNet-18 use basic blocks made up of two layers each. In contrast, larger models like ResNet-50 employ bottleneck blocks designed to enhance efficiency. These bottleneck blocks work by first compressing the input's dimensions through a 1×1 convolution, then performing deeper processing with a 3×3 convolution, and finally expanding the dimensions back to their original size with another 1×1 convolution. This structure helps to reduce the computational load while preserving the depth and capability of the network.

2.6 Vision Transformer (ViT)

Vision Transformers (ViTs) mark a significant development in the field of image processing, bringing the power of transformer architectures to computer vision. The introduction of ViTs was a groundbreaking moment, showcasing that the reliance on convolutional neural networks (CNNs) is not strictly necessary for achieving state-of-the-art results in image classification tasks.

Transformers, primarily known for their success in natural language processing, are based on self-attention mechanisms. These mechanisms allow models to weigh different parts of the input data differently, focusing on what is important. Vision Transformers apply this concept to image classification by dividing images into a sequence of patches, treating each patch as a 'word.' A critical element of ViTs is the addition of a [CLS] token that serves as the representation of an entire image, useful for classification tasks. Alongside, positional embeddings are added to maintain the spatial relationship between patches.

The ViT architecture consists of a sequence of transformer blocks that include multi-head self-attention and feed-forward neural networks. It differs from CNNs by not using convolution operations and pooling layers but rather relies on self-attention to capture global dependencies within an image. This mechanism allows ViTs to dynamically prioritize different parts of the image across different layers, adapting to the complexity of the image data without the constraints of CNNs' local receptive fields.

To elaborate on this, the ViT architecture, as depicted in Fig. 2.10, starts with the transformation of an input image into a sequence of fixed-size patches. These patches are then

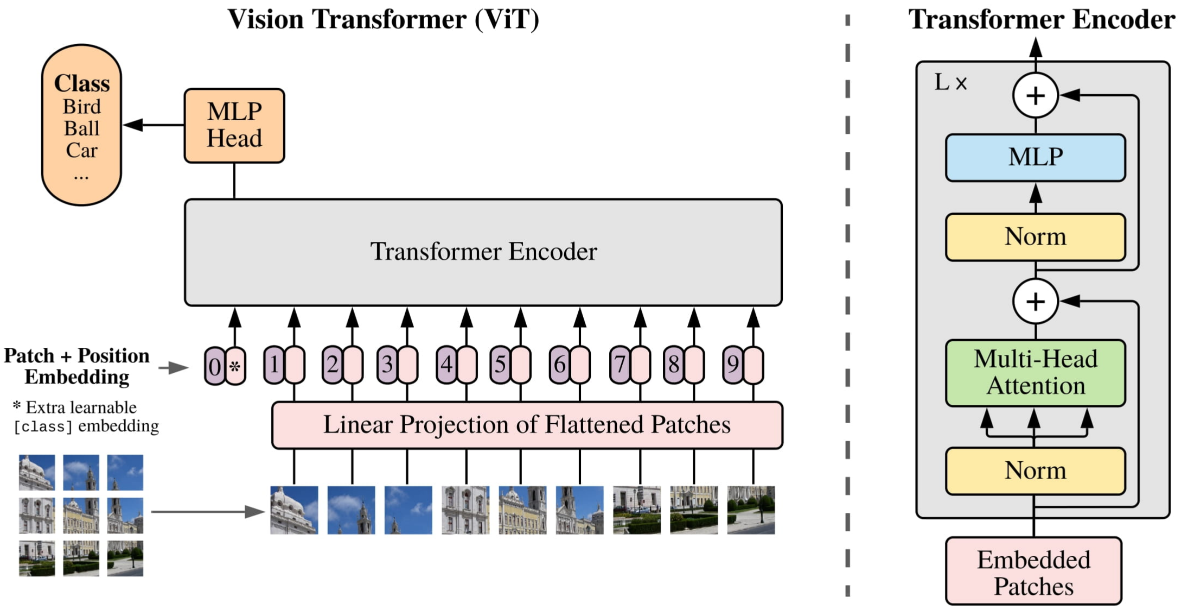

Figure 2.10: ViT architecture. Taken from [18].

linearly projected, akin to word embeddings in language models, and processed in a series of transformer blocks, each composed of multi-head self-attention and feed-forward neural network modules. The architecture abstains from utilizing traditional convolution operations and pooling layers, distinctive of CNNs, and instead employs self-attention mechanisms to ascertain global dependencies across the image. This paradigm enables ViTs to adaptively focus on various parts of the image within different layers of the architecture, adjusting to the image data's intricacies. It incorporates learned positional embeddings to retain spatial information, a step that is crucial due to the lack of inherent sequential data processing in the transformer model. Furthermore, the addition of a [CLS] token at the beginning of the sequence provides a global image representation after propagating through the transformer encoder, subsequently feeding into an MLP Head for the final classification output. The structural design of ViTs, leveraging the self-attention mechanism, allows for a dynamic prioritization of salient image features across multiple layers, thus offering an adaptable and scalable approach to image analysis without the limitations of CNNs' localized receptive fields.

One notable strength of ViTs is their scalability. Pre-trained on large datasets, ViTs can be transferred to various image recognition tasks with less computational resources than CNNs would typically require. This has been demonstrated on multiple benchmarks where ViTs attained excellent results, making them a highly efficient alternative to conventional CNNs.

Following the initial introduction of ViTs, further improvements have been proposed in the literature. Models such as Data-efficient Image Transformers (DeiT) show that transformers can also be made more data-efficient through techniques like distillation. BERT pre-training of Image Transformers (BEiT) demonstrates the power of self-supervised pre-training in improving ViT performance without reliance on labeled data. Moreover, models trained with self-supervised methods like DINO and MAE exhibit intriguing properties like object segmentation capability without explicit training for the task.

The diverse applications of ViTs include image classification, object detection, semantic segmentation, and image generation, extending to more complex scenarios such as video deepfake detection, anomaly detection, and even autonomous driving systems. For instance, in autonomous driving, dense prediction capabilities of ViTs are essential for tasks like depth estimation and understanding scene dynamics, which are crucial for safe navigation.

However, ViTs come with their own set of challenges. They generally require large amounts of data for training, which might not always be available. Moreover, their self-attention mechanisms can lead to high computational demands, especially for longer sequences of patches. Despite these challenges, the ViT model represents a significant milestone in the AI community's endeavor to develop models that can efficiently and effectively process and understand visual data.

Chapter 3

Related Work

In the field of FRB research, the majority of ML applications have primarily focused on classification problems. These fall into two main groups: distinguishing between RFI (Radio Frequency Interference) and genuine FRB signals, and differentiating between repeating and non-repeating FRBs. Since around 2016, interest in applying ML for FRB analysis has increased, but the development of more complex models has been restricted by the limited data available. To address this issue, it is common practice to generate synthetic data through various simulation methods, providing a practical solution to expand the training dataset. Despite this, some studies have successfully used ML algorithms, like Random Forest, with the relatively small number of observed FRB events. This section explores the research at the intersection of FRB studies and ML, reviewing key contributions and highlighting the approaches used in the field.

Wagstaff et al. [58] focuses on using a ML classifier, specifically Random Forests, to enhance the detection of FRBs within the VLBA's V-FASTR project [59]. The classifier aims to identify potential FRBs by categorizing events as known pulsar pulses, RFI artifacts, or possibly new discoveries. They set a 90% confidence threshold to ensure greater reliability in their classifications. It successfully filtered out 80%–90% of events, drastically cutting down the number of events that needed manual analysis. This made the process faster and improved how accurately real FRB events were spotted, highlighting how useful ML can be in astronomy.

The work by Foster et al. [19] expands on similar methodologies to Wagstaff et al., using random forest classifiers for differentiating FRBs from RFI. Their approach involved a detailed analysis of around 15,000 events, resulting in a training set for their model, which utilized about 400 features extracted from the events' characteristics, such as signal statistics from the dynamic spectrum (i.e., waterfalls) and time series. This enriched dataset and the applied features allowed for effective screening of data, significantly improving the efficiency in identifying genuine FRB candidates.

Liam Connor and Joeri van Leeuwen tackled the challenge of classifying FRBs using deep neural networks [14]. They designed a CNN that takes multiple types of input data such as dynamic spectra, pulse profiles, and DM-time data. The method mainly uses simulated FRBs added to actual survey data for training the network, covering a wide range of potential FRB characteristics. The researchers prepared two main datasets for training: one derived from CHIME Pathfinder observations [3] and another from Apertif data. Their results underscore the high efficiency of this multi-input CNN model in classifying FRB candidates, demonstrating the feasibility of using GPUs for real-time automated classification. Although initially

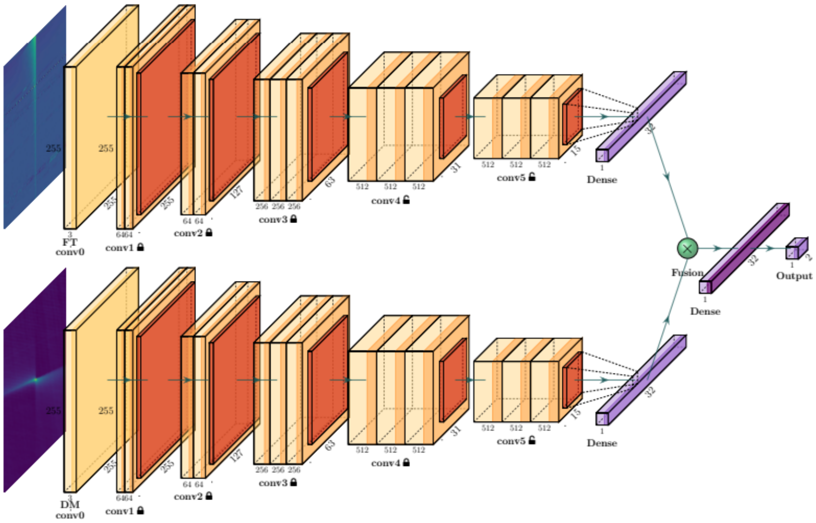

Figure 3.1: Illustration of the dual-path network architecture by Agarwal et al. employing both frequency-time and DM-time images as inputs. Key features include convolutional layers (marked in yellow) with specified output sizes, ReLU activation functions (highlighted by brown edges), and pooling layers (in orange). The architecture also integrates dense layers (in violet) and a fusion technique represented by the green circle, where the outputs of two dense layers are merged element-wise. This is followed by additional trainable layers (indicated by unlock symbols), with the initial frozen layers (lock symbols), emphasizing the layers' training status [2].

exploring the possibility of replacing traditional dedispersion methods with CNN classifiers, they noted the challenge of low signal-to-noise ratios per pixel, which might limit the effectiveness of such a substitution in current setups. However, they also highlighted the potential for future improvements as telescope technologies evolve.

In the study conducted by Agarwal et al. [2], the authors applied an innovative approach using CNNs and transfer learning to enhance the detection of FRBs. They used pre-trained CNN architectures like VGG16 [51], ResNet50 [25], and Xception [12], fine-tuning these on a dataset including simulated FRBs, real RFI instances from the Green Bank Observatory, and actual pulsar data. This process involved training two models: one using frequency-time representations to learn specto-temporal patterns of FRBs, and another using DM-time images to highlight their dispersion. The training strategy was characterized by its use of transfer learning, where the convolutional base of each model was initially frozen to prevent weights from updating, allowing only the top layers to update their weights for the new task. A key advancement in their approach was the "multiplicative fusion" technique, which merged features from both frequency-time and DM-time trained models (see Fig. 3.1). By multiplying the top layer outputs of these models element-wise, they combined extracted features of FRBs' spectro-temporal and dispersion data into a more comprehensive feature set. This step was followed by training a new set of layers for the final classification task, utilizing the combined features to more accurately identify FRBs from other types of signals. The efficacy of this approach was evident in the models' performance, which boasted accuracy and recall rates exceeding

Table 3.1: Average F2 scores and standard deviations of supervised models trained for classifying FRBs as repeaters and non-repeaters [34].

Model	F2 Mean	F2 SD
Decision tree (all features)	0.7351	0.0632
Decision tree (TB and ν)	0.7369	0.0499
Random forest	0.7821	0.0643
AdaBoost	0.7666	0.0617
LightGBM	0.7832	0.0647
XGBoost	0.7843	0.0631
SVM	0.8180	0.0483
Nearest centroid	0.7147	0.0614

99.5% on a test dataset that included real FRB events alongside RFI and noise. The models demonstrated their robustness and generalization capability by maintaining high performance across data from various telescopes and observational setups, underscoring the potential of ML in revolutionizing FRB detection.

The research “Machine learning classification of CHIME fast radio bursts – I. Supervised methods” by Lue et al. [34], employs supervised ML algorithms to analyze data from the CHIME/FRB catalogue [3], featuring 536 Fast Radio Bursts. To prepare these data, six FRBs with zero values for fluence were excluded, and each sub-burst was treated as an independent event, resulting in a total of 594 individual bursts. The algorithms, including Decision Trees, Random Forests, AdaBoost, LightGBM, XGBoost, SVM, and the Nearest Centroid Method, were trained to classify data into two groups of repeating and non-repeating events using features like fluence, peak frequency, and derived metrics such as brightness temperature and burst energy. The research further identified numerous FRBs initially categorized as non-repeaters, which the models flagged as potential repeaters. The results for each model, trained 1,000 times to ensure robustness, are summarized in Table 3.1, showing the mean and standard deviation of the F2 score across these iterations. This work underscores the power of supervised ML in enhancing our understanding of FRB origins and classifications, paving the way for more focused observational strategies.

Continuing their work on FRBs, Zhu, Ge et al. in their subsequent research, “Machine learning classification of CHIME fast radio bursts – II. Unsupervised methods” [65] used two primary categories of unsupervised ML methods: clustering and dimensionality reduction. Dimensionality reduction techniques, both linear and manifold-based, were used to transform the high-dimensional FRB data into a lower-dimensional space. For dimensionality reduction, they utilized principal component analysis (PCA), t-distributed stochastic neighbor embedding (t-SNE), and Uniform Manifold Approximation and Projection (UMAP). For clustering, the study employed hierarchical density-based spatial clustering of applications with noise (HDBSCAN) and K-means clustering to group the data points based on their similarities. The research focused on identifying as many true repeater FRBs as possible, allowing misclassifications of non-repeaters as repeaters to ensure no actual repeaters were missed. Recall, measuring how many actual repeaters the model accurately identifies, was considered more important than pre-

Table 3.2: Performance of unsupervised algorithms trained for classifying FRBs as repeaters and non-repeaters [65].

Method	**TP**	**FN**	**FP**	**TN**	**Recall**	**Precision**	**F2**
PCA + K-means	85	9	127	373	0.9042	0.4009	0.7227
t-SNE + HDBSCAN	81	13	117	383	0.8617	0.4091	0.7056
UMAP + HDBSCAN	81	13	117	383	0.8617	0.4041	0.7056

cision, which assesses the accuracy of repeater predictions. The results of their trained models are shown in Table 3.2.

In their study, Zhang et al. [64] developed a CNN designed to analyze FRB spectrogram data. The architecture they used addresses the challenges of FRB data, which are different from common computer vision datasets like MNIST [16] or ImageNet [15]. The study states that FRBs have simpler shapes than complex objects like cars or people, allowing their basic characteristics to be detected by the early layers of the neural network. However, the network needs to be complex enough to deal with a lot of pixel noise, as the goal is to identify very weak signals, sometimes even weaker than the surrounding noise. To address these challenges, they employed a residual network structure for their deep learning model. By having skip connections in the architecture, the model can bypass certain layers, which helps process simple and complex features effectively, ensuring that FRB patterns are detectable even with lots of background noise. The model also integrates batch normalization and dropout layers, which is essential for more stable training and avoiding overfitting.

For training this model, they utilized filterbank data from the Breakthrough Listen digital back-end [35], covering 4 GHz to 8 GHz, to create their dataset. They segmented these data into frames, standardized each by subtracting the mean and dividing by the standard deviation per channel to achieve zero mean and unit variance. For training and testing, they used five hours of observation data to generate around 400,000 images, half populated with simulated FRB pulses and the other half with noise and RFI. Their model achieved an accuracy of 93%, with a recall of 88% and a precision of 98% on the test set. Furthermore, They discovered 72 new pulses of FRB 121102 from the five-hour observation session.

Chapter 4

Methodology

In this section, we offer an in-depth examination of our approach to address our problem and the specific methods we utilized to develop our solutions. This begins with the process of data generation and continues with the introduction of two novel approaches for accurately estimating the DM of FRBs. Additionally, we will detail the evaluation metrics employed to measure the performance of our models.

4.1 Data

In the realm of FRB research, the data collected through telescopes serve as a crucial foundation for understanding these cosmic phenomena. Although the volume of data is currently limited due to the restricted number of known FRB sources, as shown in Table 4.1, it is steadily growing as our observational capabilities expand. However, a significant challenge in utilizing these data effectively is their inherently noisy nature. Noise, primarily from RFI, can mask the subtle signals of FRBs, necessitating sophisticated RFI mitigation techniques. Despite these efforts, complete noise elimination remains elusive, and a certain degree of background noise persists.

Additionally, a portion of the FRB data available to researchers comes pre-processed in a dedispersed form, which, under specific circumstances, might not be suitable for all analytical methods. While it is technically practical to re-disperse this data for our training purposes, the dual challenges of persistent noise and the limited dataset size prevent the development of robust models. Given these challenges, we have decided to employ synthetic data generation techniques for our ML models.

4.1.1 Data Generation

In this part, we go over the challenge of generating FRB data, a task complicated by numerous influencing factors. To generate synthetic data, we have expanded upon a software package available on GitHub[1], which is distributed under the GPL-2.0 license. Our enhancements and additional features substantially improve its functionality, aiming for simulations that closely approximate real observations. Although our customized software now facilitates simulations

---

[1] See: https://github.com/liamconnor/single_pulse_ml

Telescope	**Number of Reported FRB Sources**
CHIME	633
ASKAP	60
Parkes	40
UTMOST	18
GBT	15
Arecibo	14
LPA	11
VLA	4
FAST	5
GMRT	4
MeerKAT	3
SRT	3
EFFELSBERG	3
DSA	3
APERTIF	6
STOCKERT	1
TIANLAI	1
Total	**813**

Table 4.1: Number of Reported FRB Sources through various telescopes according to FRB-STATS [53].

across a variety of telescopes used for the collection of FRB data, as detailed in Table 4.2, the simulated data set used in this thesis is in the format and frequency range of the CHIME telescope. That is, our synthetic FRBs have a bandwidth of 400 MHz centered at 600 MHz.

To begin with, we initialize several variables randomly, such as the DM, scattering timescale, and fluence, according to probability distributions to emulate the randomness observed in real FRB data. This approach ensures that our synthetic data closely mirrors the diverse characteristics of actual observations. To further align our synthetic FRB data with the observational capabilities of various telescopes, we calculate the operational time resolution and the maximum DM that can be represented within the constraints of our simulations. The operational time resolution of the telescope, t_{samp} (in ms), is computed as follows:

$$t_{\text{samp}}\ (\text{ms}) = \left[\frac{1.5}{\nu_{\text{ref}}\,(\text{GHz})}\right]\left(\frac{\nu_{\text{max}} - \nu_{\text{min}}}{\nu_{\text{ref}}}\right) \tag{4.1}$$

indicating the resolution at which the telescope operates, and where ν_{min}, ν_{max} and ν_{ref} are, respectively, minimum, maximum, and centre frequencies of the telescope's receiver bandwidth (in GHz for this equation; see Table 4.2). On the other hand, the maximum DM, DM_{max}, that our synthetic data can accommodate, utilizing N_{time} time bins, is determined by the equation:

$$\text{DM}_{\text{max}} = N_{\text{time}}\left(\frac{t_{\text{samp}}}{a}\right)\left(\frac{1}{\nu^2_{\text{min}}} - \frac{1}{\nu^2_{\text{max}}}\right)^{-1} \tag{4.2}$$

with a defined in equation (2.2). This enables us to simulate FRBs with dispersion measures up to DM_{max}, determined by the use of N_{time} time bins. This choice establishes a balance between

Telescope	ν_{min} **(MHz)**	ν_{max} **(MHz)**	ν_{ref} **(MHz)**
CHIME	400	800	600
Arecibo (ALFA)	1225	1525	1375
Arecibo (L-Wide)	1150	1730	1440
GMRT (Band 3)	550	850	700
Effelsberg	1200	1740	1470
GBT (Prime)	290	396	343
GBT (L-Band)	1150	1730	1440
GBT (S-Band)	1730	2600	2165
GBT (C-Band)	3950	7800	5075
GBT (X-Band)	7800	11600	9700
FAST	1050	1450	1250
LOFAR	110	240	175

Table 4.2: Telescope spectral information for data generation with ν_{min}, ν_{max} and ν_{ref} are, respectively, minimum, maximum, and centre frequencies of the telescope's receiver bandwidth. Our synthetic data are generated in the CHIME format.

capturing detailed features and maintaining computational efficiency in our synthetic dataset. For our simulated CHIME-like dataset we have $t_{\text{samp}} = 1.67$ ms, from equation (4.1), and $0 \leq \text{DM} \leq 87\ \text{cm}^{-3}$ pc, from equation (4.2) using 512×1024 data arrays (i.e., $N_{\text{freq}} \times N_{\text{time}}$).

Following this initialization, we focus on the generation of a background noise array that has the same dimensions of the signal, represented by N_{freq}, for the number of frequency bins, and N_{time} (see Fig. 4.1). This foundational step is pivotal for simulating a realistic observational background, which is later combined with the signal to simulate a more realistic burst. The noise matrix is populated with values drawn from a Gaussian distribution characterized by a mean of zero and a standard deviation of one. Concurrently, we create a list of frequencies that starts at ν_{min} and ends at ν_{max}, with N_{freq} evenly spaced frequencies in between to simulate the signal across the frequency range we are studying.

The next phase of the simulation involves the calculation of the pulse's effective width, which helps us understand how much the signal spreads out and gets wider. Initially, the scattering timescale is set to zero, but in later steps we incorporate scattering dynamics. For determining the effective width, dispersion smearing is computed as follows:

$$\Delta t_{\text{DM}}\,(\text{ms}) = 2a\,\text{DM}\left(\frac{\Delta\nu}{\nu_{\text{ref}}^{3}}\right), \tag{4.3}$$

where $\Delta\nu$ is the frequency resolution. This expression is a derivative of the general dispersion relation (see equation 2.1) that quantifies the temporal delay induced by signal dispersion within the interstellar medium. Also, it should be mentioned that the DM is chosen from a uniform distribution within the specified range. The effective width of the pulse, however, encompasses more than just dispersion smearing; it also includes the intrinsic width of the signal which is drawn from a Gaussian distribution and the resolution of the observing instrument. Having all these values, the effective width is calculated using equation (2.6). We next determine when the signal arrives at the current frequency channel compared to the reference

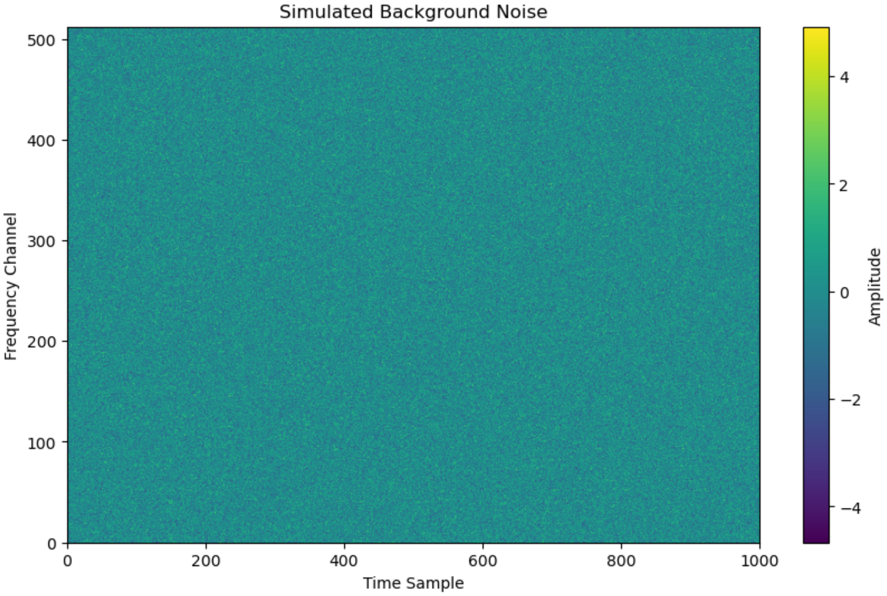

Figure 4.1: Illustration of randomly generated background noise .

frequency ν_{ref}, i.e., at the centre of the bandwidth, using equation (2.1) while also incorporating the further delay caused by the sub-burst slope law given in equation (2.7). We then have for the arrival time (in ms) at frequency ν

$$t_{\text{arrival}}(\nu) = a\,\text{DM}\left(\frac{1}{\nu^2} - \frac{1}{\nu_{\text{ref}}^2}\right) - t_{\text{w}}\left(\frac{\nu - \nu_{\text{max}}}{A\,\nu_{\text{ref}}}\right), \tag{4.4}$$

where we set $A = 0.1$ for our simulations. By dividing both the arrival time and the pulse's effective width by the time resolution, we can pinpoint exactly when and for how long the burst appears in this channel.

Moving to the next step in the simulation process, after determining the arrival time index denoted as $t_{\text{arrival\_index}}$ and the effective width index as $t_{\text{w\_index}}$ for the pulse based on the time resolution, we focus on the creation of the pulse profile. The Gaussian profile serves as the basic shape for our pulse due to its prevalent use in modeling various physical phenomena, including signal shapes in astrophysical simulations. We thus use:

$$g(t) = \exp\left[-\frac{(t - t_{\text{arrival\_index}})^2}{2\,t_{\text{w\_index}}^2}\right], \tag{4.5}$$

for the temporal shape of the pulse. In this formula, $t_{\text{arrival\_index}}$ is when the pulse reaches its peak, and $t_{\text{w\_index}}$ determines the spread of the pulse; a larger $t_{\text{w\_index}}$ results in a broader and flatter pulse.

Building on the previously discussed scattering and propagation effects (see Sec. 2.1), the simulation progresses to integrate these effects into the signal's overall profile. For the scattering effect, the timescale τ_s is a function of the signal's frequency, as outlined in equation (2.4)

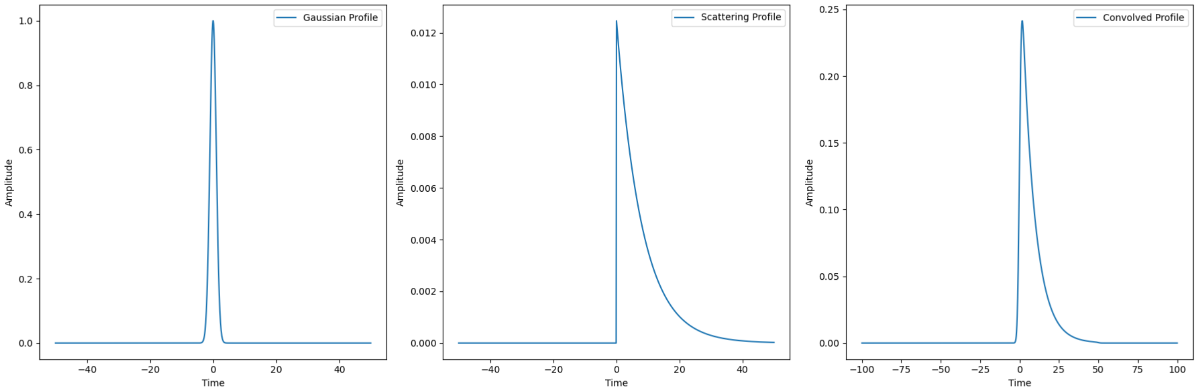

Figure 4.2: Illustration of the convolution process between a Gaussian profile and a scattering profile. Please note the different scale of the time axis for the last plot.

(setting $n = 4$ in it), where τ_0 is drawn from a log-uniform distribution. The frequency dependency implies that lower frequencies experience a more pronounced scattering. The scattering kernel is modeled through an exponential decay:

$$h(t) = \frac{1}{\tau_s} \exp\left(-\frac{t}{\tau_s}\right) \tag{4.6}$$

with the time t extends from 0 to $N_{\mathrm{time}}/2$. The integral of $h(t)$ over this interval is then normalized to unity, since the effect of scattering is the result of the convolution of $h(t)$ with $g(t)$. This ensures a uniform treatment of scattering effects across the simulated signal bandwidth. The convolution process mathematically combines the two profiles to produce a composite signal that embodies both the natural shape of the astrophysical source and the effects of scattering, as shown in Fig. 4.2.

Another effect we need to take into account is scintillation. Incorporating scintillation into the simulation of FRB signals involves creating an amplitude modulation across the frequency band to reflect the natural fluctuations caused by the interstellar medium. The mathematical formulation in equation (2.5) is used for the scintillation envelope in the simulation process. The parameter N_{scint} sets the scale or intensity of the scintillation effect. To capture the wide range of scintillation scales observed in nature, N_{scint} is generated through a log-uniform distribution between 0.001 and 7. This approach ensures the simulation spans a broad spectrum of scintillation intensities, from very subtle to significantly pronounced, favoring milder effects to reflect their more frequent occurrence in natural scintillation patterns. In equation (2.5) ν represents the frequencies at which the FRB signal is being simulated, and ν_{ref} is the reference frequency used to normalize the frequency dependence of the scintillation effect. A random phase ϕ_{scint} is introduced to ensure variability in the scintillation pattern across different simulations, enhancing the realism of the modeled effect.

The formula employs a cosine function to model the amplitude fluctuations caused by scintillation. The frequency functionality $(\nu_{\mathrm{ref}}/\nu)^2$ implies that the scintillation effect decreases with increasing frequency, aligning with observational data that suggest scintillation is more pronounced at lower frequencies. The choice of a cosine function, along with the specific formulation for the frequency dependence and the inclusion of a random phase, allows for the

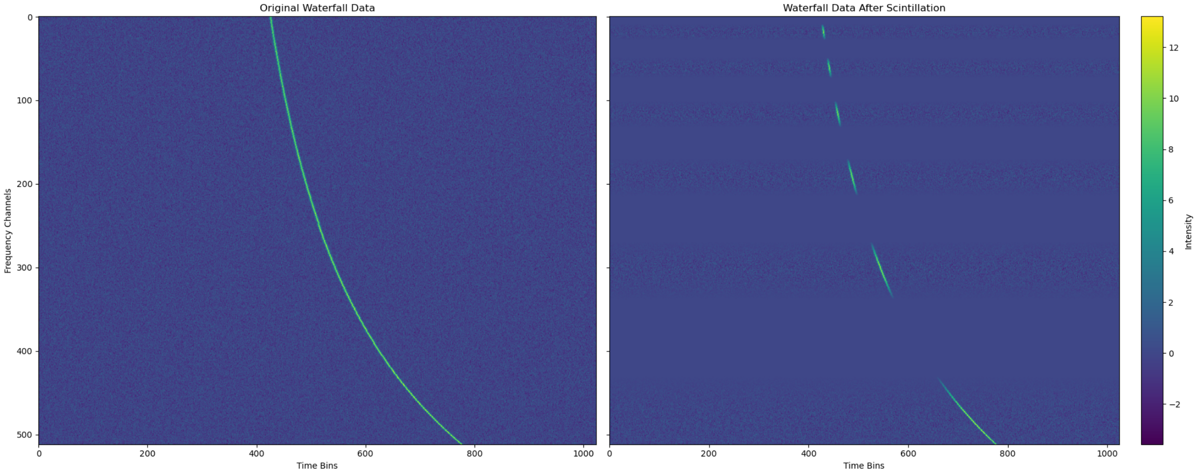

Figure 4.3: Illustration of the effect of scintillation applied to a signal.

simulation of complex, naturally occurring scintillation patterns in a computationally efficient manner. The N_{scint} value, chosen from a log-uniform distribution, controls the strength of the twinkling effect is – higher values make the signal vary more dramatically across different frequencies, while the method of selection ensures a bias towards more commonly observed, milder scintillation effects. This mathematical model captures the essence of scintillation, allowing for the realistic portrayal of its impact on simulated FRB signals (see Fig. 4.3).

As the last steps in our simulation process, we perform several adjustments and refine the pulse profiles further, ensuring they accurately represent the complex nature of FRB signals as detected by radio telescopes. We normalize the pulse profile against its maximum amplitude and the standard deviation of the noise data, aligning the signal strength with ambient noise levels and maintaining realistic signal-to-noise ratios. This normalization is crucial for ensuring the pulse does not overwhelmingly dominate the background noise, similar to real observational data. Next, we scale the normalized profile by the fluence of the pulse, representing the total energy distributed across the frequency band. The fluence is picked randomly from a uniform distribution. This step adjusts the overall intensity of the signal, reflecting the different energy levels.

Finally, we incorporate adjustments for frequency-dependent amplitude variations using a spectral index which is also chosen from a uniform distribution. This step simulates how the pulse's strength varies across different frequencies with respect to a reference frequency, capturing the frequency-dependent behavior of FRB signals. These adjustments based on frequency are crucial for accurately simulating how the pulse's strength changes because of its spectral characteristics and the properties of the space it travels through. Once all the steps are completed, the pulse profiles are merged with the background noise to create synthetic FRB data ready for our project as shown in Fig. 4.4.

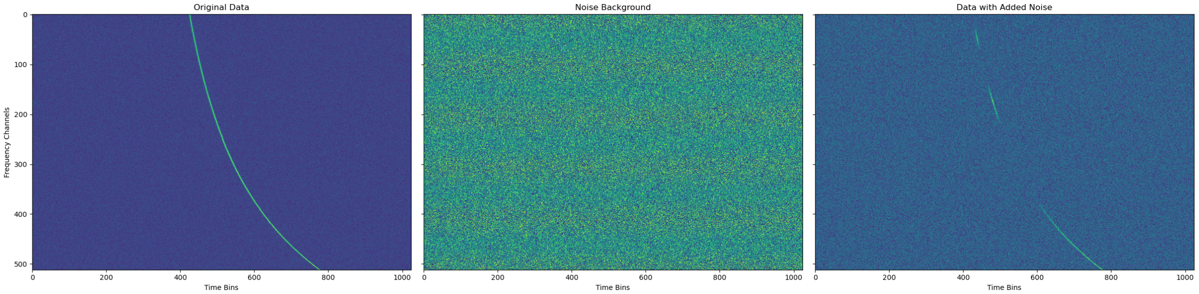

Figure 4.4: An illustration of the generated data from the described process is shown here, with the noise level intentionally amplified for clearer visualization.

4.2 Model Training

In this section, we outline the development of two new approaches for estimating DM values. Model training was conducted on the Narval cluster available through the Digital Research Alliance of Canada, equipped with an A100 GPU, 128 GB of RAM, and an AMD Milan 7413 CPU. We used a 3070Ti GPU, 32 GB RAM, and an Intel 12700k CPU for the analyses after training.

4.2.1 Baseline CNN Model

The baseline model is designed as a CNN featuring a series of convolutional layers followed by fully connected layers. At the core of the architecture are seven convolutional layers, each equipped with batch normalization to stabilize learning and improve convergence times. Additionally, the ReLU activation function is applied after each convolutional layer, introducing non-linearity and allowing the model to learn complex patterns more effectively. The convolutional layers progressively increase in depth, starting from 32 filters in the first layer and doubling in number through successive layers, culminating in 2048 filters in the final convolutional layer. This design allows the model to capture a wide range of features from the input data, from simple patterns in the early layers to more complex structures in the deeper layers.

Between each convolutional layer, except for the 3rd and 6th, max pooling operations are applied to reduce the spatial dimensions of the feature maps. This decreases the computational load and enhances the model's ability to generalize. After the convolutional layers, the model employs fully connected layers. These layers integrate the high-level features identified by the convolutional layers to produce an output. This output quantifies the estimated DM. The sequence of fully connected layers begins with 16,384 neurons in the first layer and progressively halves in size.

During the hyper-parameter tuning process, we experimented with various network architectures, starting with a baseline configuration of four convolutional layers and three fully connected layers, each comprising 512 nodes. Our objective was to optimize the architecture to achieve the best possible performance without overfitting the data. To further refine our model, we incorporated batch normalization layers, which significantly sped up the training process—achieving comparable results in just 40 epochs as opposed to the initial 60 epochs required without these layers.

Layer (Type)	**Configuration**
Conv2D-1 + BN-1 + ReLU MaxPool2d-1	1 input, 32 outputs, kernel=3x3, stride=1, padding=1 kernel size=2, stride=2
Conv2D-2 + BN-2 + ReLU MaxPool2d-2	32 inputs, 64 outputs, kernel=3x3, stride=1, padding=1 kernel size=2, stride=2
Conv2D-3 + BN-3 + ReLU	64 inputs, 128 outputs, kernel=3x3, stride=1, padding=1
Conv2D-4 + BN-4 + ReLU MaxPool2d-3	128 inputs, 256 outputs, kernel=3x3, stride=1, padding=1 kernel size=2, stride=2
Conv2D-5 + BN-5 + ReLU MaxPool2d-4	256 inputs, 512 outputs, kernel=3x3, stride=1, padding=1 kernel size=2, stride=2
Conv2D-6 + BN-6 + ReLU	512 inputs, 1024 outputs, kernel=3x3, stride=1, padding=1
Conv2D-7 + BN-7 + ReLU MaxPool2d-5	1024 inputs, 2048 outputs, kernel=3x3, stride=1, padding=1 kernel size=2, stride=2
Flatten	7 x 7 x 2048 = 100,352
FC-1 FC-2 FC-3 FC-4 FC-5	100,352 inputs, 16384 outputs 16384 inputs, 8192 outputs 8192 inputs, 4096 outputs 4096 inputs, 2048 outputs 2048 inputs, 1 output

Table 4.3: Architecture of Baseline CNN Model.

The role of dropout layers in our experiments presented a trade-off between model generalization and the accuracy of DM estimations. Initially, we tested dropout probabilities of 0, 0.05, 0.1, and 0.2 to assess their impact on the model's learning capability and generalization. Although higher dropout rates improved generalization, they also reduced the model's ability to learn intricate patterns effectively, as evidenced by a decrease in estimation accuracy. Additionally, in our data, each pixel value does not carry meaning by itself, and there exists a significant spatial dependency between pixels [64]. This inherent characteristic of our data further discouraged the use of dropout, as removing pixels randomly could disrupt these critical spatial relationships, undermining the learning process. Given these observations, we decided against employing dropout layers within the fully connected segments of our architecture. This decision was informed by the analysis of model performance across the various dropout configurations and the specific nature of our input data.

Furthermore, When loading each pulse profile for training, we generate a noise matrix matching the dimensions of the pulse profile. Elements of this matrix are randomly drawn from a uniform distribution between 0 and 1. Incorporating this noise directly into the pulse profiles helps our model become more robust to noise and aids in mitigating overfitting, enhancing the model's generalization capability. This technique ensures that even though the underlying pulse profile remains the same across epochs, the noise overlay is different each time. Afterward, we normalize the data and scale the images to a range of 0–255.

Finally, we focused on minimizing the Mean Absolute Error (MAE) loss function, where accurate numeric prediction is the goal of training. To minimize this loss function, we utilized the Adam optimizer because of its efficiency in handling large datasets and complex architectures. The combination of MAE loss and Adam optimizer allowed us to iteratively adjust our model parameters to reduce prediction errors, aiming to achieve the lowest possible cost across our training dataset. In this process, we conducted experiments to identify the optimal learning rate from a range of values, including 0.01, 0.001, 0.0001, 0.00001, 0.000001, and 0.0000001. Our findings revealed that the smallest learning rate of 0.0000001 yielded the best results. This is because larger learning rates tended to cause the model to diverge from finding the optimal solution. The smaller learning rate enabled more gradual and precise adjustments to the model's parameters, having a smoother convergence to the optimal solution and significantly improving the accuracy of our estimations.

4.2.2 DM Estimation utilizing Transfer Learning

Transfer Learning has proven to be a powerful method in ML tasks, especially when there are limitations such as insufficient data for training models from scratch, or constraints on time and computational resources. This approach allows a pre-trained model to be adapted for a more specific, related task. This adaptation generally involves fine-tuning the model's deeper layers responsible for capturing more specific features while retaining and utilizing the early layers that extract general features. By fine-tuning pre-trained models, researchers and practitioners can achieve better results with fewer data, lower computational resources, and a shorter amount of time.

This is particularly effective with CNNs because they are designed to recognize basic features like edges in their early layers and gradually identify more complex patterns such as textures in deeper layers [61]. Considering that FRBs have relatively simple shapes, these basic and intermediate learned features by large pre-trained models could be significantly useful for our estimation task. These facts, coupled with the success of residual networks in classifying FRBs as discussed in related works, led us to select ResNet50 for our task. Originally trained on the ImageNet dataset to classify images into 1,000 categories, ResNet50 has learned to recognize a broad range of basic image features [25]. By fine-tuning ResNet50 for DM estimation, we leverage its capability to detect essential features relevant to our specific task.

Furthermore, the emergence of transformer models and self-attention mechanisms has shown promising results across various domains, including Natural Language Processing (NLP) and Computer Vision tasks. Their ability to capture long-range dependencies and complex patterns makes them an interesting choice for FRB analysis. Thus, we also decided to explore the potential of ViT for DM estimation. By fine-tuning ViT, which has been pre-trained on the ImageNet dataset also, we aim to understand the advantages and potential limitations of applying transformer architectures to the domain of FRBs.

Both models were fine-tuned with the same dataset as the baseline model with a different preprocessing approach. The preprocessing of spectrogram data involves several steps, making the data ready for training with ResNet50 and ViT. After reading the data from generated files, we introduce a new noise matrix to the data in each iteration or epoch to increase the robustness of our models. This training phase approach helps the model identify signals even with varying levels and patterns of noise. After adding noise, we scale the data values between 0 and 1 using

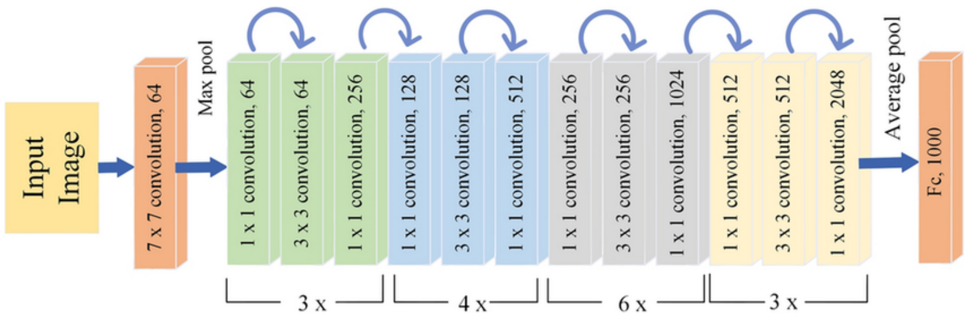

Figure 4.5: An illustration of Resnet50 architecture [10].

min-max normalization through

$$X_{\text{scaled}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}. \tag{4.7}$$

This step ensures that all data points are within a consistent range, essential for the neural network to process them effectively. The data are then converted to a 0–255 scale, making them compatible with additional image processing libraries. Depending on the model we are fine-tuning, the images are resized to match the input requirements: 224×224 pixels for ResNet50 and 384×384 pixels for ViT. Since both models were originally designed for three-channel (RGB) input, we replicate the grayscale image across three channels. Finally, the images are normalized using specific mean and standard deviation values ([0.485, 0.456, 0.406] for the mean and [0.229, 0.224, 0.225] for the standard deviation). This normalization aligns our data with the distribution expected by the pre-trained models.

For fine-tuning the pre-trained ResNet50 model for DM estimation, we customized it to fit our regression task rather than its original classification objective. Specifically, the model's final classification layer including 1000 neurons corresponding to different classes was replaced with a linear layer having only one neuron to output a continuous DM value. This modification aligns the model's output with the requirements of our regression task. We particularly focused on the model's last convolution layers, identified as Layer 4 and Layer 5. In Fig. 4.5, Layer 4 is indicated by the gray color and consists of 6 Residual Networks, each with three layers, while Layer 5, highlighted in yellow, comprises 3 Residual Networks, also with three layers each. This segment of the model was made trainable, enabling it to learn FRB-specific features from our dataset. In contrast, the earlier layers, which are trained to capture basic image characteristics such as edges and textures, were kept frozen meaning their weights were not changed during the training process. The fine-tuning process took 40 epochs, utilizing a learning rate of 0.0001. This chosen learning rate facilitates gradual and precise adjustments, ensuring the model converges to optimal performance without overshooting. Through this fine-tuning process, the ResNet50 model was effectively repurposed for DM estimation, leveraging its deep learning capabilities to extract meaningful insights from FRB spectrogram data.

As we mentioned earlier, the second model that we fine-tuned for our task was ViT. To align the model architecture with our regression goal we replaced its classification head with a linear layer, enabling it to predict continuous DM values. The ViT encoder, highlighted in gray in Fig. 2.10, has 12 layers, as shown on the right side of the figure. This encoder serves as the core component of the model. It processes patched images to generate a feature map that encapsulates the image's characteristics. We unfroze the last 7 layers to allow the model to adjust its weights to learn FRB-specific features, while the earlier layers remained frozen to preserve learned generic features. The fine-tuning process is performed over 40 epochs with a 10^{-5} learning rate.

For both the ResNet50 and ViT models, we employed the L1 Loss function from the Pytorch framework, also known as Mean Absolute Error (MAE), as our cost function. This choice is motivated by the nature of our task which is a regression problem where we try to predict the exact target value.

4.3 Model Evaluation

For evaluating the performance of our models on the test dataset, we employ two commonly used metrics: Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). These metrics provide a comprehensive view of how well our models predict DM values compared to actual observations. These are defined with

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{4.8}$$

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|, \tag{4.9}$$

where y_i is the actual DM, $\hat{y}_i$ is the predicted DM, and n is the number of observations in the test dataset.

RMSE is especially valuable for emphasizing larger errors in predictions due to its calculation method, which involves squaring the differences before calculating the average and taking the square root of the result. This sensitivity to larger errors makes RMSE crucial for applications like FRB analysis where accuracy can be critical. A lower RMSE indicates that the model's predictions are not only close on average to the true DM values but also that extreme deviations are uncommon. On the other hand, MAE provides a straightforward measure of the average magnitude of errors, treating all deviations equally without giving undue weight to larger errors. This metric quantifies the typical deviation of our model's predictions from the actual values, offering an intuitive understanding of prediction accuracy.

Employing both RMSE and MAE for evaluation provides a balanced understanding of our model's performance in estimating DM on new data. This dual metric approach ensures that our estimations are not only precise on average but also robust, minimizing large deviations that could affect the reliability of FRB dispersion measure predictions.

Chapter 5

Results & Discussion

In this section, we present the results of our trained models for the task of DM estimation from two-dimensional frequency-time images of FRBs (i.e., waterfalls). Each model – ResNet50, ViT, and a baseline CNN – was trained or fine-tuned through extensive trial and error to optimize performance. To ensure robustness in our findings, we report outcomes from the top five training iterations for each model, including their average performance. We employed RMSE and MAE as our key metrics to evaluate and compare the effectiveness of the models in predicting DM values.

5.1 Dataset

The models were trained on a synthetic dataset consisting of 79,000 samples generated in the manner described in Sec. 4.1.1. This dataset was split into training and validation sets, with 30% of the data reserved for validation and the remaining 70% used for training. The validation set was used for hyper-parameter tuning during model development. Additionally, to evaluate the models' performance on new, unseen data, we generated 8,000 additional data points using the same parameters that defined the original train and validation sets. These new data points serve as a test set to evaluate and ensure the models' robustness in predicting dispersion measures. Both the training and validation sets follow a similar distribution, as illustrated in Figure 5.1. This figure displays the distribution of DM value, which is the target value our models aim to predict. Consistent data distribution across training and validation phases is important to ensure that the models learn to generalize well from the training data to unseen data.

5.2 Baseline CNN Model

The performance of our baseline CNN model serves as a reference point for evaluating the advanced capabilities of ViT and ResNet50 models. This model incorporates standard convolutional layers without the architectural innovations found in ResNet or ViT. To ensure a comprehensive evaluation, we report the results of five training iterations, each accomplished over 40 epochs.

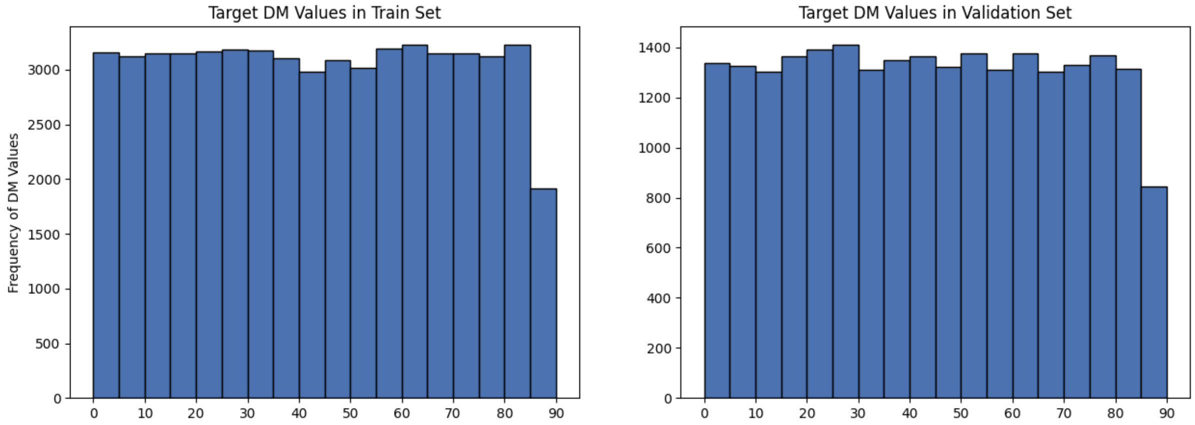

Figure 5.1: Illustration of DM distribution (i.e., $0 \leq \text{DM} \leq 87$ cm^{-3} pc) for training and validation sets.

The performance of the baseline CNN model can also be assessed through the distribution of absolute differences between the predicted and actual DM values. For example, for Run 4 in Table 5.1, which has the lowest MAE and RMSE on the test data, the maximum observed difference was 2.682 cm^{-3} pc, while the average difference was approximately 0.1815 cm^{-3} pc and the standard deviation was 0.1666 cm^{-3} pc, suggesting a generally tight clustering around the mean value. To visually and statistically depict these differences, we constructed a histogram (shown in Figure 5.2) that categorizes the differences into several bins with intervals of 0.5 cm^{-3} pc, starting from 0 and extending up to the maximum difference observed.

The histogram illustrates that a significant majority of the predictions (7,655 out of 8,000) fall within a 0.5 cm^{-3} pc difference, underscoring the model's accuracy. As the difference range increases, the frequency of larger differences declines, with only 316 predictions differing by 0.5 to 1.0 cm^{-3} pc, and even fewer for higher intervals – 21 predictions in the 1.0 to 1.5 cm^{-3} pc range, 7 in the 1.5 to 2.0 cm^{-3} pc range, and 1 prediction in the 2.5 to 3.0 cm^{-3} pc range. There were no predictions within the 2.0 to 2.5 cm^{-3} pc. This distribution confirms the baseline CNN model's competent performance in DM estimation, as most predictions align closely with the

Table 5.1: Results of the baseline CNN Model over five runs. Shown are the MAE on the training, validation, and test data sets, as well as the RMSE on the test data with averages over the five runs.

Run	Train MAE (cm^{-3} pc)	Validation MAE (cm^{-3} pc)	Test Loss MAE (cm^{-3} pc)	Test Loss RMSE (cm^{-3} pc)
1	0.1894	0.1922	0.1924	0.2562
2	0.2276	0.1747	0.2057	0.2688
3	0.2371	0.2162	0.2181	0.2971
4	0.2126	0.1860	0.1815	0.2464
5	0.2207	0.1989	0.2021	0.2837
Average	0.2174	0.1936	0.1999	0.2704

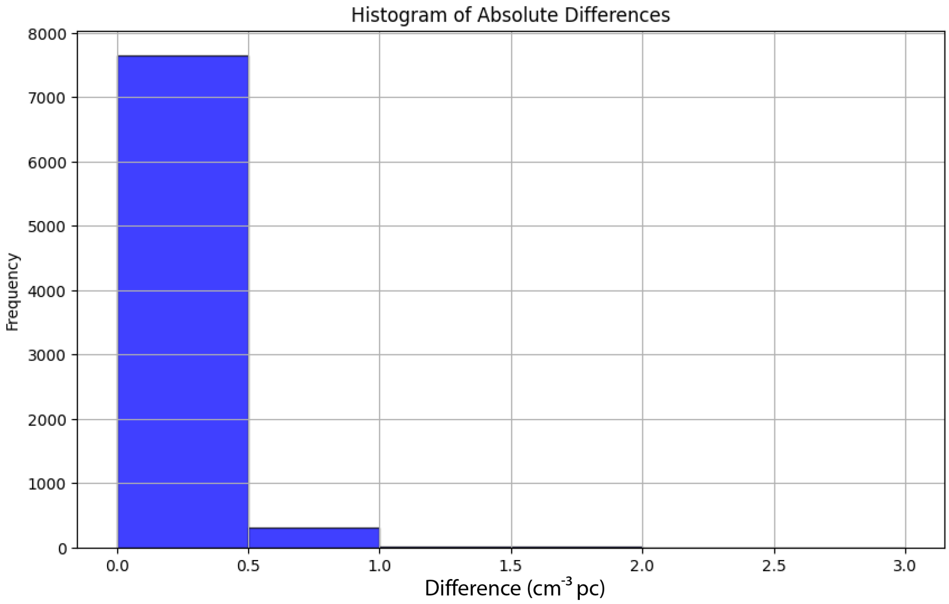

Figure 5.2: Histogram of absolute differences for DM predictions by the baseline CNN model.

actual data, demonstrating its effectiveness and reliability.

5.3 Vision Transformers

The performance of the ViT model specifically fine-tuned for DM estimation was evaluated over five training runs. The primary goal was to assess the model's ability to accurately predict continuous DM values. The results of this evaluation are summarized in Table 5.2. Each training run of the ViT model showcases the robustness of the fine-tuning strategy, with the model consistently achieving a low MAE across training, validation and test datasets.

In Figure 5.3, we present two visualizations to illustrate the performance of the ViT model.

Table 5.2: Results of the ViT Model over five runs. Shown are the MAE on the training, validation, and test data sets, as well as the RMSE on the test data with averages over the five runs.

Run	Train MAE (cm^{-3} pc)	Validation MAE (cm^{-3} pc)	Test Loss MAE (cm^{-3} pc)	Test Loss RMSE (cm^{-3} pc)
1	0.2036	0.2350	0.2346	0.3131
2	0.2247	0.2349	0.2365	0.3212
3	0.1966	0.2390	0.2394	0.3284
4	0.2158	0.2302	0.2315	0.3107
5	0.2299	0.2176	0.2153	0.3099
Average	0.21412	0.23134	0.23146	0.3166

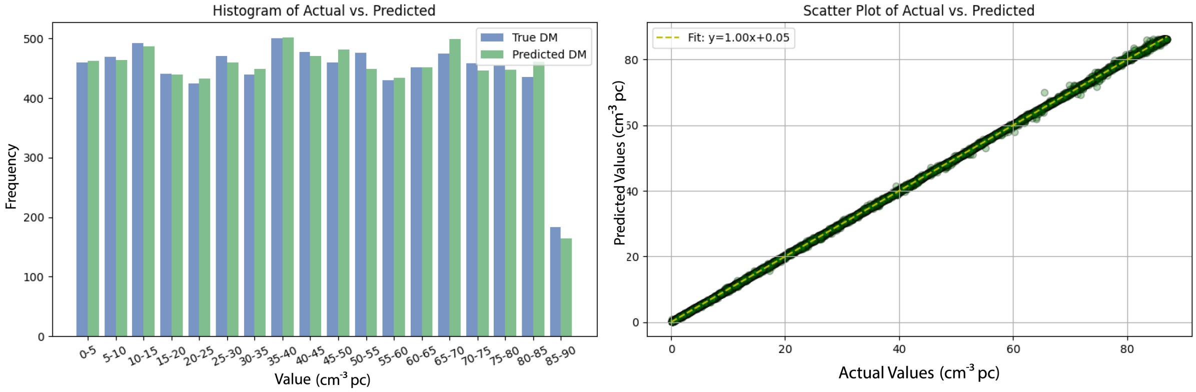

Figure 5.3: Performance visualization of the fine-tuned ViT model. *Left:* Histogram showing the distribution of actual versus predicted DM values. *Right:* Scatter plot with a fitted line ($y = x + 0.05$).

The left panel displays a histogram comparing the actual versus predicted DM values, highlighting the distributional alignment and frequency of predictions across predefined value bins. This histogram serves as a robust measure of the model's ability to capture the overall trend and variance in the DM values. The right panel shows a scatter plot with a fitted line of best fit ($y = x + 0.05$), which indicates a nearly ideal proportional relationship between the predicted and actual values, albeit with a slight overestimation of 0.05 cm^{-3} pc. This positive intercept suggests a systematic bias where the model, on average, predicts slightly higher DM values than the actual ones.

To quantify the prediction errors, we calculated the absolute differences between predicted and actual DM values for Run 5 in Table 5.2 and plotted these differences in Fig. 5.4. This histogram provides insight into the error distribution, revealing the frequency of various error magnitudes. Smaller bars at higher difference values indicate that the model generally predicts with high accuracy, with most discrepancies being relatively minor. The majority of predictions (7,383 out of 8,000) exhibit a difference of 0.0 to 0.5 cm^{-3} pc, indicating that the model is highly accurate for the bulk of its predictions. As the difference increases, the frequency of occurrences significantly drops, with 524 predictions showing differences between 0.5 to 1.0 cm^{-3} pc, and smaller counts as the difference further widens. Only a minimal number of predictions (93) exhibit differences greater than 1.0 unit, with the occurrences tapering off to nearly none as the difference approaches and exceeds 3.0 cm^{-3} pc. This demonstrates that the model's predictions are generally quite close to the actual values, with very few large errors. The maximum observed difference is approximately 4.64 cm^{-3} pc, indicating the largest error in the predictions. The average difference across all predictions stands at approximately 0.22 cm^{-3} pc, which signifies that, on average, the model's predictions are closely aligned with the actual values. Additionally, the standard deviation of the differences is about 0.22 cm^{-3} pc, reflecting a moderate spread of errors around the mean.

Furthermore, It should be mentioned that we encountered significant challenges in our experiments to fine-tune the ViT huge-size model. Despite the theoretical advantages offered by this large-scale pre-trained model, practical limitations became evident during the fine-tuning process. The extensive computational resources required resulted in each epoch consuming ap-

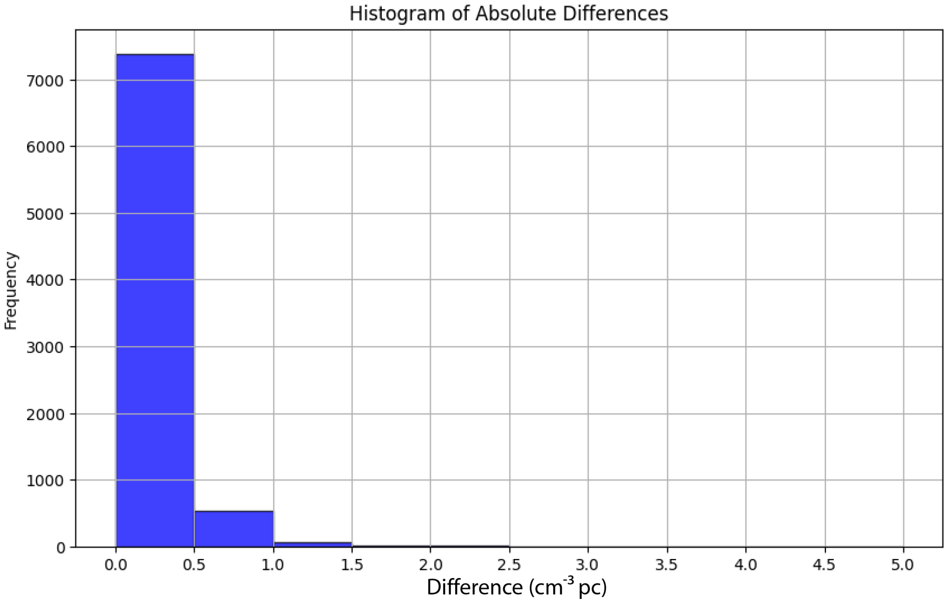

Figure 5.4: Histogram of residuals for DM predictions by the ViT model.

proximately six hours, making the training process prohibitively time-consuming given our resource constraints. Also, initially designed to perform optimally with vast and diverse datasets, the model did not adapt well to the relatively smaller dataset specific to our task. The best performance achieved was a MAE of 1.7 cm^{-3} pc on the training and 1.9 cm^{-3} pc on the test sets. These results fell short of expectations considering the model's capabilities and the computational effort involved. Recognizing these limitations, we opted to switch to a smaller, more manageable base model, which was better suited to our data's scale and computational resources. This transition allowed us to continue refining our approach while aligning more closely with the practical realities of our dataset and computational framework.

5.4 Resnet50

In this section, we detail the performance of the fine-tuned ResNet50 model. As with the ViT model, the ResNet50 model underwent a series of optimizations through trial and error to ensure peak performance. The ResNet50 model's adaptation for DM estimation involved replacing its final classification layer with a linear regression layer to directly predict continuous DM values. This adjustment allows the model to leverage its deep, residual learning framework effectively for our regression problem. Over the course of five training runs, the ResNet50 demonstrated strong performance across both the validation and test datasets. A summary of these results is presented in Table 5.3, highlighting its robustness and precision in DM estimation.

Figures similar to those presented for the ViT model are used to visually represent the ResNet50's performance. Figure 5.5 illustrates the distribution of actual versus predicted DM values and their correlation, providing a visual assessment of the model's accuracy and systematic biases.

Table 5.3: Results of the Resnet50 Model over five runs. Shown are the MAE on the training, validation, and test data sets, as well as the RMSE on the test data with averages over the five runs.

Run	Train MAE (cm^{-3} pc)	Validation MAE (cm^{-3} pc)	Test Loss MAE (cm^{-3} pc)	Test Loss RMSE (cm^{-3} pc)
1	0.2535	0.1822	0.1793	0.2599
2	0.1905	0.1607	0.1600	0.2088
3	0.1918	0.1767	0.1782	0.2267
4	0.2354	0.1700	0.1715	0.2348
5	0.1835	0.1769	0.1753	0.2258
Average	0.2109	0.1733	0.1728	0.2312

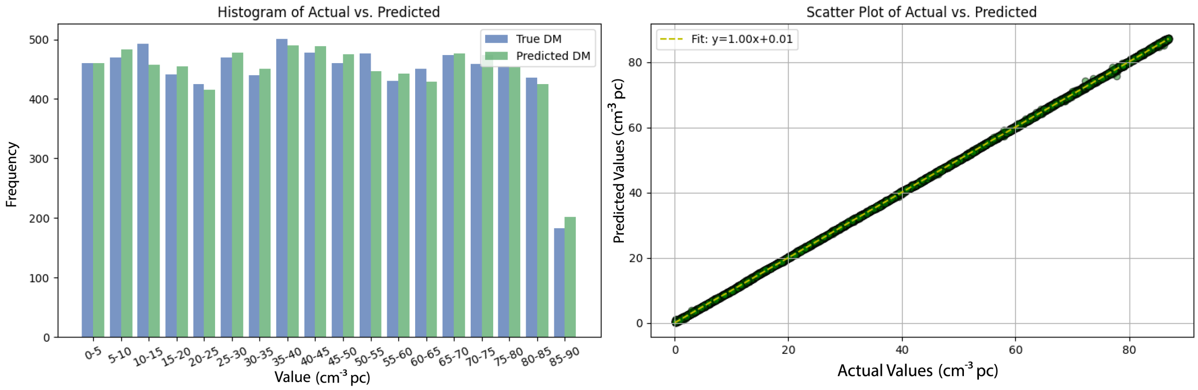

Figure 5.5: Performance visualization of the fine-tuned ResNet50 model. *Left:* Histogram showing the distribution of actual versus predicted DM values. *Right:* Scatter plot illustrating the relationship between actual and predicted values, fitted with a line ($y = x + 0.01$).

The performance of the fine-tuned ResNet50 model also could be quantified through the distribution of absolute differences between predicted and actual DM values. For Run 2 in Table 5.3, the maximum observed difference is 2.110 cm^{-3} pc, while the average difference is 0.1600 cm^{-3} pc, with a standard deviation of 0.1341 cm^{-3} pc, indicating a generally tight clustering around the mean value. To provide a visual and statistical understanding of these differences, we constructed a histogram (shown in Figure 5.6) that categorizes the differences into several bins with intervals of 0.5 cm^{-3} pc, starting from 0 up to the maximum difference. The results show that a significant majority of the predictions (7,826 out of 8,000) fall within a 0.5 cm^{-3} pc difference. As the range increases, the frequency of larger differences sharply decreases, with only 165 predictions differing by 0.5 to 1.0 cm^{-3} pc, and a further decrease for higher intervals–7 predictions in the 1.0 to 1.5 cm^{-3} pc range, 1 in the 1.5 to 2.0 cm^{-3} pc range, and 1 prediction in the 2.0 to 2.5 cm^{-3} pc range. This histogram analysis demonstrates that the fine-tuned ResNet50 model performs well in estimating DM values, with the vast majority of predictions closely aligning with the actual data.

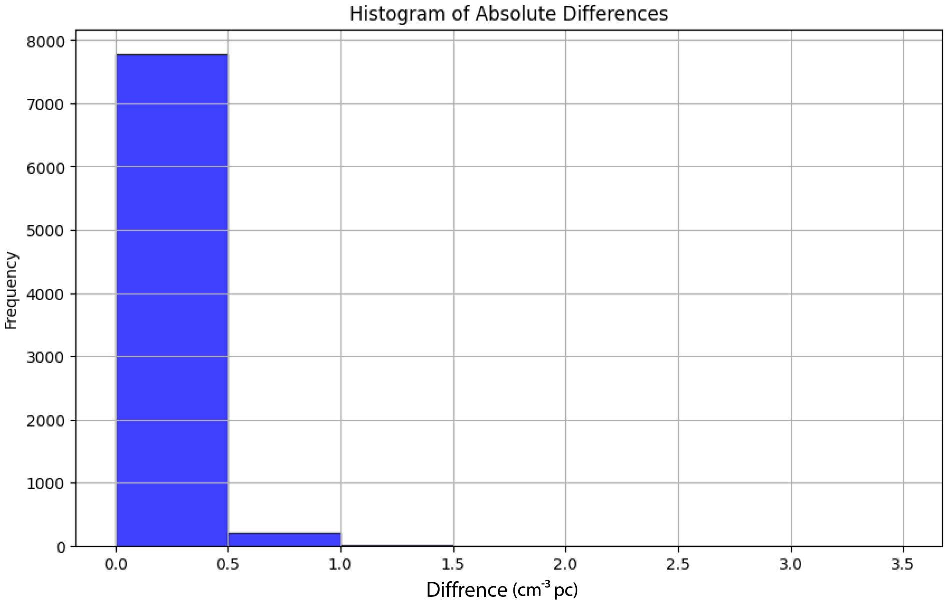

Figure 5.6: Histogram of Residuals for DM Predictions..

5.5 Discussion

It is essential to highlight the inference times and practicality of integrating these models into a real-time streaming framework. We measured the inference speed of our models on an NVIDIA GTX–3070Ti, evaluating them with a test dataset of 8000 images using a batch size of 128. The time resolution of the generated images is 0.0016 seconds over 1024 bins, totaling 1.634 seconds.

The fastest model, ResNet50, processed images at a remarkable rate, capable of handling approximately 72 images per second, translating to one image every 0.013 seconds. This speed allows for covering roughly 118 seconds of observation time. In contrast, our baseline CNN model required 811 seconds, and the Vision Transformer took 616.2 seconds for the same dataset, significantly slower than ResNet50.

When compared to traditional dedispersion methods like those employed by HEIMDALL [4], which scales with the number of DM trials – processing about 3 seconds of observation per second with 1200 DM trials [64] – our network's inference capability is faster. This significant acceleration in processing speed underlines the feasibility of deploying our deep learning models, particularly the fine-tuned ResNet50, in real-time FRB analysis pipelines, potentially transforming how we handle the streaming of astronomical data.

The accuracy of our models is another critical factor. The fine-tuned ResNet50 model showed the best performance with averages of 0.1728 cm^{-3} pc for MAE and 0.2312 cm^{-3} pc for RMSE, indicating its effectiveness in accurately predicting DM values. In comparison, the ViT recorded an MAE of 0.23146 cm^{-3} pc and an RMSE of 0.3166 cm^{-3} pc, while the baseline CNN model achieved an MAE of 0.1999 cm^{-3} pc and an RMSE of 0.2704 cm^{-3} pc. Surprisingly, the baseline CNN model also performs well in terms of accuracy, exceeding expectations for such a straightforward architecture.

The performance of the ViT might be further enhanced with additional training data or by exploring different variations of the ViT architecture. Although its current accuracy and inference speed are the least optimal among the three, the adaptability of the ViT framework suggests that with proper tuning and scaling, its performance could significantly improve, potentially matching or surpassing the other models. This adaptability makes it a promising candidate for future enhancements.

It is also important to verify if our current accuracy in DM estimation is sufficient for effectively studying the spectro-temporal characteristics of FRBs, especially through the measurement of the sub-burst slope law. We can do so by considering equation (4.4) for the arrival time of an FRB as a function of frequency. More precisely, since the two terms on the right-hand side of this equation are, respectively, for the delay due to dispersion and that related to the sub-burst slope law [46] we must ensure that the latter dominates after dedispersion. In other words, the error in DM estimate $\Delta\mathrm{DM}$ must be such that

$$\Delta\mathrm{DM} < \frac{\nu_{\mathrm{ref}}^2 t_{\mathrm{w}}}{2aA}, \tag{5.1}$$

where we approximated $|\nu - \nu_{\mathrm{ref}}| \ll \nu_{\mathrm{ref}}$. However, it is found that on average $\nu_{\mathrm{ref}} t_{\mathrm{w}} \approx 1.5\,\mathrm{ms\,GHz}$ [8] such that

$$\Delta\mathrm{DM} \lesssim 0.75 \frac{\nu_{\mathrm{ref}}}{aA}. \tag{5.2}$$

Using a scaling $A \approx 0.08$ [8], we find the constraint $\Delta\mathrm{DM} \lesssim 1.4$ at 600 MHz, which eases considerably at higher frequency (e.g., $\Delta\mathrm{DM} \lesssim 9$ at 4 GHz). Our stated performance, evaluated through the MAE and RMSE metrics, therefore readily meets the requirements needed for the study of the spectro-temporal characteristics of FRBs.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we investigated the application of ML, particularly deep learning, for estimating the DM of FRBs. We developed three distinct models for this regression task, each demonstrating impressive results. While serving as a proof of concept, our work substantiates that ML can effectively handle a limited range of DM values, yielding excellent outcomes.

Our analysis confirmed that all three models fall under the error threshold discussed earlier, illustrating their high reliability. The fine-tuned ResNet50 model, in particular, achieved an inference speed 60 times faster than the traditional brute-force methods currently employed, showcasing the potential of deep learning models to revolutionize real-time astronomical data analysis.

We discussed related works, highlighting various ML approaches applied to the classification of FRBs. Inspired by these studies, our baseline model and the fine-tuned ResNet50 were adapted from methodologies originally developed for classification tasks. Additionally, we explored the potential of the ViT for this regression problem. Although ViTs typically require more data than ResNets and CNNs, our results indicate promising directions for further refinement with increased data availability.

Significant enhancements were made to the data generation package used in previous studies by Connor et al. [14], modifying it to our specific needs. Modifications included the implementation of a sub-burst slope law, adjustments to the distribution of random variables to better reflect realistic scenarios, corrections of previously unnoticed errors, and expanded support for different telescopes and resolutions.

Put all these together, the advancements demonstrated in this thesis highlight the substantial benefits of applying ML to the analysis of FRBs, in terms of both performance and accuracy. These findings pave the way for future research that could eventually lead to deployable models capable of handling a broader range of conditions in real-world settings.

6.2 Future Work

The models developed in this thesis currently cover a DM range of $0 - 87\,\mathrm{cm^{-3}\,pc}$. Although this range is limited, our results demonstrate that deep learning models are capable of learning

FRB characteristics to predict DMs accurately. To make these models more applicable to real-world scenarios, future work should expand the range of DM values in the training data.

Further research could explore the use of more complex models such as ResNet101 and ResNet152, which are designed to capture intricate patterns and might perform well over extended ranges. However, before progressing to these larger models, it is advisable to maximize the potential of the current models through extensive fine-tuning.

The ViT base variant showed promising results even with limited data. Expanding the dataset could improve its performance, and exploring other variants such as ViT-B\32, ViT-L\16, and ViT-L\32 might yield further improvements. These models could be fine-tuned to employ their capabilities to the specific challenges of DM estimation.

Addressing the detection and mitigation of Radio Frequency Interference (RFI) and noise in the received data is crucial. Developing a model to effectively filter out the RFI could significantly improve the subsequent DM estimation steps and enhance overall data quality for further analysis.

With advancements in ML models for image generation, which are capable of producing detailed and varied outputs, there is potential to adapt these tools for generating synthetic FRB data. This approach could help overcome the issue of data scarcity in FRB research.

Finally, expanding the scope of DM estimation to accommodate data from multiple telescopes could be beneficial for comprehensive offline analyses. However, for real-time streaming data analysis, optimizing a single model to deliver accurate DM estimates for each telescope separately might prove more practical. This focused approach could streamline the processing pipeline, enhancing the responsiveness and accuracy of live data analysis systems.

By addressing these areas, future research can significantly advance the application of ML in the field of FRBs, pushing the boundaries of what is currently achievable with automated analysis techniques.

Bibliography

[1] Hervé Abdi and Lynne J. Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010.

[2] Devansh Agarwal, Kshitij Aggarwal, Sarah Burke-Spolaor, Duncan R Lorimer, and Nathaniel Garver-Daniels. Towards deeper neural networks for fast radio burst detection. *arXiv preprint arXiv:1902.06343*, 2019.

[3] Mandana Amiri, Bridget C Andersen, Kevin Bandura, Sabrina Berger, Mohit Bhardwaj, Michelle M Boyce, PJ Boyle, Charanjot Brar, Daniela Breitman, Tomas Cassanelli, et al. Erratum:"the first chime/frb fast radio burst catalog"(2021, apjs, 257, 59). *The Astrophysical Journal Supplement Series*, 264(2):53, 2023.

[4] B. R. Barsdell, M. Bailes, D. G. Barnes, and C. J. Fluke. Accelerating incoherent dedispersion: Accelerating incoherent dedispersion. *Monthly Notices of the Royal Astronomical Society*, 422(1):379–392, March 2012.

[5] Benjamin R Barsdell, Matthew Bailes, David G Barnes, and Christopher J Fluke. Accelerating incoherent dedispersion. *Monthly Notices of the Royal Astronomical Society*, 422(1):379–392, 2012.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[7] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[8] Katie Brown, Mohammed A. Chamma, Fereshteh Rajabi, Aishwarya Kumar, Hosein Rajabi, and Martin Houde. Validating the sub-burst slope law: a comprehensive multi-source spectro-temporal analysis of repeating fast radio bursts. *Mon. Not. R. Astron. Soc.*, 529(1):L152–L158, March 2024.

[9] Ricardo J. G. B. Campello, Davoud Moulavi, and Jörg Sander. Density-based clustering based on hierarchical density estimates. *Proceedings of the 17th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2013)*, pages 160–172, 2013.

[10] Anthony Ashwin Peter Chazhoor, Edmond S. L. Ho, Bin Gao, and Wai Lok Woo. A review and benchmark on state-of-the-art steel defects detection. *SN Computer Science*, 5(1):114, Dec 2023.

[11] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.

[12] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[13] Chime/Frb Collaboration, Bridget C Andersen, Kevin Bandura, Mohit Bhardwaj, PJ Boyle, Charanjot Brar, Daniela Breitman, Tomas Cassanelli, Shami Chatterjee, Pragya Chawla, et al. Sub-second periodicity in a fast radio burst. *Nature*, 607(7918):256–259, 2022.

[14] Liam Connor and Joeri van Leeuwen. Applying deep learning to fast radio burst classification. *The Astronomical Journal*, 156(6):256, 2018.

[15] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[16] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[17] Fengqiu Adam Dong, Chime/Frb Collaboration, et al. Chime/frb detection of a bright radio burst from sgr 1935+ 2154. *The Astronomer's Telegram*, 15681:1, 2022.

[18] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[19] Griffin Foster, Aris Karastergiou, Golnoosh Golpayegani, Mayuresh Surnis, Duncan R Lorimer, Jayanth Chennamangalam, Maura McLaughlin, Wes Armour, Jeff Cobb, David H E MacMahon, Xin Pei, Kaustubh Rajwade, Andrew P V Siemion, Dan Werthimer, and Chris J Williams. ALFABURST: a commensal search for fast radio bursts with Arecibo. *Monthly Notices of the Royal Astronomical Society*, 474(3):3847–3856, 11 2017.

[20] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[21] V Gajjar, APV Siemion, DC Price, CJ Law, D Michilli, JWT Hessels, S Chatterjee, AM Archibald, GC Bower, C Brinkman, et al. Highest frequency detection of frb 121102 at 4–8 ghz using the breakthrough listen digital backend at the green bank telescope. *The Astrophysical Journal*, 863(1):2, 2018.

[22] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[26] Muhammad Tausif Irshad, Muhammad Adeel Nisar, Xinyu Huang, Jana Hartz, Olaf Flak, Frédéric Li, Philip Gouverneur, Artur Piet, Kerstin M. Oltmanns, and Marcin Grzegorzek. Sensehunger: Machine learning approach to hunger detection using wearable sensors. *Sensors*, 22(20), 2022.

[27] I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag New York, 2nd edition, 2002.

[28] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[29] Muhammad Yaseen Khan, Abdul Qayoom, Muhammad Nizami, Muhammad Shoaib Siddiqui, Shaukat Wasi, and Khaliq-Ur-Rahman Raazi Syed. Automated prediction of good dictionary examples (gdex): A comprehensive experiment with distant supervision, machine learning, and word embedding-based deep learning techniques. *Complexity*, 09 2021.

[30] Petros Pavlakis Konstantinos Topouzelis, Vassilia Karathanassi and Demetrius Rokos. Potentiality of feed-forward neural networks for classifying dark formations to oil spills and look-alikes. *Geocarto International*, 24(3):179–191, 2009.

[31] S. R. Kulkarni. Dispersion measure: Confusion, Constants & Clarity. *arXiv e-prints*, page arXiv:2007.02886, July 2020.

[32] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[33] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[34] Jia-Wei Luo, Jia-Ming Zhu-Ge, and Bing Zhang. Machine learning classification of chime fast radio bursts – i. supervised methods. *Monthly Notices of the Royal Astronomical Society*, 518(2):1629–1641, November 2022.

[35] David H. E. MacMahon, Danny C. Price, Matthew Lebofsky, Andrew P. V. Siemion, Steve Croft, David DeBoer, J. Emilio Enriquez, Vishal Gajjar, Gregory Hellbourg, Howard Isaacson, Dan Werthimer, Zuhra Abdurashidova, Marty Bloss, Joe Brandt, Ramon Creager, John Ford, Ryan S. Lynch, Ronald J. Maddalena, Randy McCullough, Jason Ray,

Mark Whitehead, and Dave Woody. The breakthrough listen search for intelligent life: A wideband data recorder system for the robert c. byrd green bank telescope. *Publications of the Astronomical Society of the Pacific*, 130(986):044502, feb 2018.

[36] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[37] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[38] Tom M Mitchell. Machine learning, 1997.

[39] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, 2 edition, 2018.

[40] Anthony J. Myles, Robert N. Feudale, Yang Liu, Nathaniel Woody, and Steven D. Brown. An introduction to decision tree modeling. *Journal of Chemometrics*, 18, 2004.

[41] W. S. Noble. What is a support vector machine? *Nature Biotechnology*, 24(12):1565–1567, 2006.

[42] Thais Mayumi Oshiro, Pedro Santoro Perez, and José Augusto Baranauskas. How many trees in a random forest? In *IAPR International Conference on Machine Learning and Data Mining in Pattern Recognition*, 2012.

[43] Tayyip Ozcan and Alper Basturk. Transfer learning-based convolutional neural networks with heuristic optimization for hand gesture recognition. *Neural Computing and Applications*, 31(12):8955–8970, Dec 2019.

[44] Emily Petroff, JWT Hessels, and DR Lorimer. Fast radio bursts. *The Astronomy and Astrophysics Review*, 27(1):4, 2019.

[45] Z Pleunis, Daniele Michilli, CG Bassa, JWT Hessels, A Naidu, BC Andersen, P Chawla, E Fonseca, A Gopinath, VM Kaspi, et al. Lofar detection of 110–188 mhz emission and frequency-dependent activity from frb 20180916b. *The Astrophysical Journal Letters*, 911(1):L3, 2021.

[46] Fereshteh Rajabi, Mohammed A. Chamma, Christopher M. Wyenberg, Abhilash Mathews, and Martin Houde. A simple relationship for the spectro-temporal structure of bursts from FRB 121102. *Mon. Not. R. Astron. Soc.*, 498(4):4936–4942, November 2020.

[47] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[48] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[49] Alessio Sclocco, Joeri van Leeuwen, Henri E Bal, and Rob V van Nieuwpoort. Real-time dedispersion for fast radio transient surveys, using auto tuning on many-core accelerators. *Astronomy and computing*, 14:1–7, 2016.

[50] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[51] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, 2015.

[52] MP Snelders, K Nimmo, JWT Hessels, Z Bensellam, LP Zwaan, P Chawla, OS Ould-Boukattine, F Kirsten, JT Faber, and V Gajjar. Detection of ultra-fast radio bursts from frb 20121102a. *Nature Astronomy*, 7(12):1486–1496, 2023.

[53] Apostolos Spanakis-Misirlis and Cameron L. Van Eck. Frbstats: A web-based platform for visualization of fast radio burst properties. *The Open Journal of Astrophysics*, 6, February 2023.

[54] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[55] D. Thornton, B. Stappers, M. Bailes, B. Barsdell, S. Bates, N. D. R. Bhat, M. Burgay, S. Burke-Spolaor, D. J. Champion, P. Coster, N. D'Amico, A. Jameson, S. Johnston, M. Keith, M. Kramer, L. Levin, S. Milia, C. Ng, A. Possenti, and W. van Straten. A Population of Fast Radio Bursts at Cosmological Distances. *Science*, 341(6141):53–56, July 2013.

[56] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[57] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., 1995.

[58] Kiri L. Wagstaff, Benyang Tang, David R. Thompson, Shakeh Khudikyan, Jane Wyngaard, Adam T. Deller, Divya Palaniswamy, Steven J. Tingay, and Randall B. Wayth. A machine learning classifier for fast radio burst detection at the vlba. *Publications of the Astronomical Society of the Pacific*, 128(966):084503, jun 2016.

[59] Randall B. Wayth, Walter F. Brisken, Adam T. Deller, Walid A. Majid, David R. Thompson, Steven J. Tingay, and Kiri L. Wagstaff. V-fastr: The vlba fast radio transients experiment. *The Astrophysical Journal*, 735(2):97, June 2011.

[60] Huo Yingge, Imran Ali, and Kang-Yoon Lee. Deep neural networks on chip - a survey. In *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 589–592, 2020.

[61] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization, 2015.

[62] Barak Zackay and Eran O Ofek. An accurate and efficient algorithm for detection of radio bursts with an unknown dispersion measure, for single-dish telescopes and interferometers. *The Astrophysical Journal*, 835(1):11, 2017.

[63] Bing Zhang. The physical mechanisms of fast radio bursts. *Nature*, 587(7832):45–53, 2020.

[64] Yunfan Gerry Zhang, Vishal Gajjar, Griffin Foster, Andrew Siemion, James Cordes, Casey Law, and Yu Wang. Fast radio burst 121102 pulse detection and periodicity: a machine learning approach. *The Astrophysical Journal*, 866(2):149, 2018.

[65] Jia-Ming Zhu-Ge, Jia-Wei Luo, and Bing Zhang. Machine learning classification of chime fast radio bursts – ii. unsupervised methods. *Monthly Notices of the Royal Astronomical Society*, 519(2):1823–1836, December 2022.

Curriculum Vitae

Name:	*Hosein Rajabi*
Education:	University of Tabriz Tabriz 2016 - 2021 B.Sc.
	University of Western Ontario London, ON 2022 - 2024 M.Sc.
Related Work Experience:	Teaching Assistant The University of Western Ontario 2022 - 2024
Publication:	Katie Brown, Mohammed A. Chamma, Fereshteh Rajabi, Aishwarya Kumar, Hosein Rajabi, Martin Houde, *”Validating the sub-burst slope law: a comprehensive multisource spectro-temporal analysis of repeating fast radio bursts,”* *Monthly Notices of the Royal Astronomical Society*, 529(1), pp. 152–158, March 2024.