Neuroevolutionary Techniques in Fast Radio Burst Classification

A Comparative Study

by

Hariprasad SV

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Bachelor of Science (Honours)
Physics

at
St. Stephen's College
University of Delhi
2023

This dissertation is guided by:

Dr. Geetanjali Sethi[†]
Dr. Annu Malhotra[†],
[†]Associate Professor, Department of Physics,
St. Stephen's College,Delhi

Neuroevolutionary Techniques in Fast Radio Burst Classification

A Comparitive study

Hariprasad SV

Abstract

Fast radio bursts (FRBs) are enigmatic astrophysical events characterized by short durations of pulses ranging from milliseconds and flux densities spanning 0.1 to 100 Jy. Classifying these events from raw intensity data involves employing various analysis techniques, such as de-dispersion for trial DM (dispersion measure) values and filtering out radio frequency interference (RFI). Since the discovery of the prototype source by Lorimer et al. in 2007, the number of identified FRB sources has been steadily increasing.

In recent years, ground-based surveys, including the CHIME (Canadian Hydrogen Intensity Mapping Experiment) project, have generated vast amounts of data, necessitating the development of efficient data analysis pipelines. This work aims to explore the novelty of utilizing Genetic Neural Network algorithms for the classification of FRB data and, by extension, other raw data obtained from diverse astrophysical surveys. The complex nature of FRB data classification makes it an ideal candidate for investigating the potential of data classification in various astrophysical surveys. An inherent advantage of using evolutionary algorithms is their parallelizability and faster runtimes, providing an opportunity for more efficient analysis. This novel approach offers a creative avenue for addressing the challenges associated with FRB data classification.

By employing Genetic Neural Network algorithms, this study aims to enhance our understanding of FRBs and contribute to the development of effective data classification methodologies in astrophysics.

Contents

List of Figures

List of Tables

Chapter 1

Introduction to Fast radio bursts

Fast radio burst (*FRB*) is an enigmatic astronomical transient phenomena of millisecond duration and flux densities in the range of $0.1 - 100$ Jy. *FRB*s belong to a much broader of class of *fast radio transients*, commonly referring to describe millisecond duration pulses that are produced by a coherent form of non thermal radio emission. The search for *fast radio transients* began in the late 1960s. With the discovery of the first Pulsar (Pulsating radio source) in 1967[1]. Pulsars exhibited a periodic behaviour, which from the analysis of the flux densities revealed a sharp rotation period. The origin of Pulsars was then determined to be the radio emissions from highly magnetized rotating neutron stars.

The development and expansion of radio surveys enhanced the search for fast radio transients. The open source availability of radio data also led to the widespread analysis and thus leading to the discovery of *Rotating Radio transients* (*RRATs*); highly intermittent galactic pulsars detected in single pulse searches. and *Giant Pulses* (*GPs*), pulses of extragalactic origin with highly intense individual pulses[2].

The existing searches eventually led upto the discovery of the first prototype *FRB* signature by *Lorimer et al.* while analysing the archival data from the *Megallanic clouds*[3]. Further four other sources were discovered by Thronton et al,[4]. The main characteristic of these signatures was the high frequency dispersion which cannot come from the Milky Way alone. Thus proving to be extra-galactic in origin.

A number of FRBs has been detected ever since, A web catalogue by Petroff et al.[5][1], contains FRB detections up to June 2020. In 2021 CHIME released a catalogue of 536 new FRBs observed from the CHIME survey.[6] .

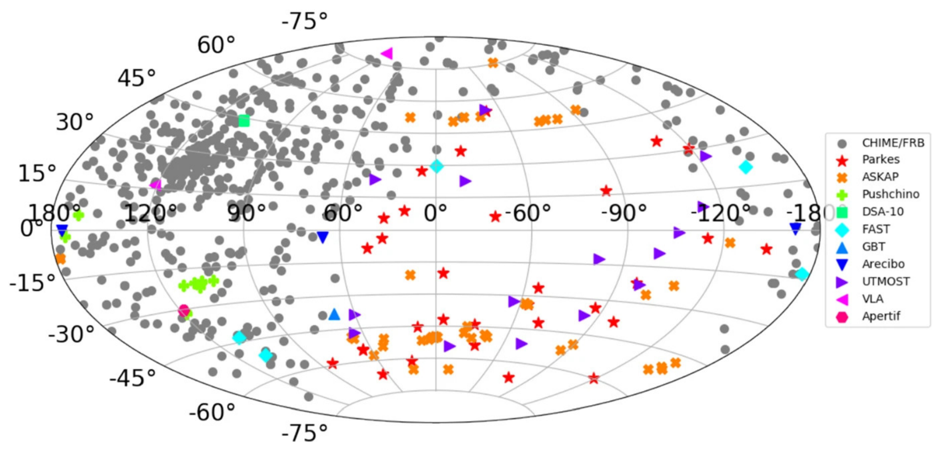


Figure 1.1: Distribution of observed FRBs till 2020

CHIME's new 1024 beam survey has significantly enriched the detection rates. CHIME currently searches 400 - 800 MHz range, which facilitates detections of newer single pulse profiles.

1.1 Relevant terms and explanations

1.1.1 Interstellar medium

The interstellar medium (ISM) consists of dust and gas in the galaxy. As a radio wave propagates through ISM, it gets attenuated and undergoes other propagation effects.

1.1.2 Dispersion

The refractive index of the propagation medium; plasma is frequency dependent. This is called dispersion. The refractive index μ can be related to the frequency ν as,

$$\mu = \sqrt{\left(1 - \frac{\nu_p^2}{\nu^2}\right)} \tag{1.1}$$

[1]www.frbcat.org

Where the plasma frequency ν_p is directly dependent on the number density.

$$\nu_p = \sqrt{\frac{n_e e^2}{\pi m_e}}$$

where m_e, n_e, e are electron mass, number density and charge respectively. Due to dispersion, a time delay is observed between higher frequency waves and lower frequency waves.

$$t = \left(\int_0^d \frac{dl}{v_g}\right) - \frac{d}{c} \tag{1.2}$$

where d is the distance from earth, from this we obtain the following integral.

$$t = \frac{e^2}{2\pi m_e c \nu^2} \int_0^d n_e dl - \frac{d}{c} \tag{1.3}$$

The integral over the electron number density n_e over the distance from the source to observer is termed as the **Dispersion Measure** (DM). DM is measured in units of $\frac{pc}{cm^3}$. DM is used as a direct substitute for source distance measurement. Using DM and evaluating the expression, we can obtain the following relation relating the dispersion delay of two frequencies ν_1 and ν_2 as,

$$\Delta t = 4.15 \times 10^6 DM \left(\frac{1}{\nu_1^2} - \frac{1}{\nu_2^2}\right) \tag{1.4}$$

Different DM models are used for calculations, NE2001 and YMW16 models are widely used in galactic and intergalactic distance calculations.

1.1.3 Dynamic Spectra

Dynamic spectra provide valuable insights into the temporal morphological structure of radio events. They are typically represented as waterfall plots, which depict the variation of signal intensity over time and frequency. By plotting the intensity as a function of time and frequency, these plots capture the changing characteristics of the radio event. In the case of radio astronomy, dynamic spectra are particularly useful for studying FRBs. When analyzing dynamic spectra, one can extract further information by dedispersing the data. Dedispersion involves taking the effective mean values along the columns of the frequency-time plot, resulting in a dedispersed pulse profile. This process helps to reveal the intrinsic pulse shape by compensating for the dispersion effects caused by the interstellar medium. While pulse profiles in some cases exhibit a Gaussian shape due to it's sharp event detection, FRBs are known to exhibit a wide variety of pulse profiles, showcasing their diverse nature. A waterfall plot and its corresponding dedispersed pulse profile obtained from the CHIME catalogue[6].is shown below.

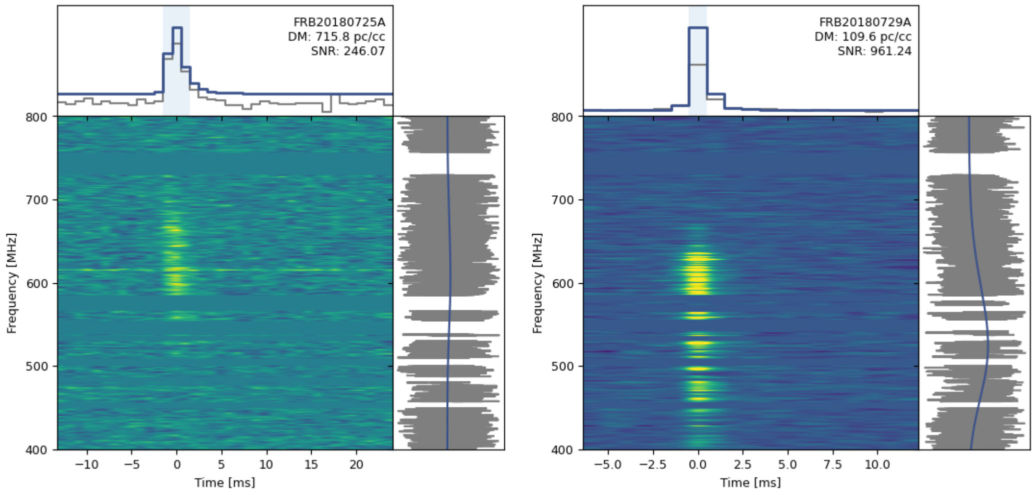


Figure 1.2: FRB20180725A & FRB20180729A - Dynamic Spectra along with dedispered pulse profile[6]

1.2 Brief overview of detection pipelines

Before delving into the classification, a brief understanding of the modern single pulse classification pipelines is required. A summary is given below. These pipelines are regularly used in FRB surveys to detect and classify FRBs according to their various features (repeatng, non repeating, DM etc)

1.2.1 Collection of radio data

The radio data is collected as raw voltage data. This analog data is converted into digital format by using analog to digital converters. This is then channelized using a spectrometer.[7] This channelized data is generally known as a filterbank.

1.2.2 Mitigation of RFI

The obtained data is often contaminated with radio noise from background, major sources include GPS, interference with broadcasting systems, routers etc. To minimize the effect of RFI, multiple techniques are utilized. Two most commonly used techniques are as follows.

- Thresholding: Reduction of RFI based on the assumption that the noise follows Gaussian noise profiles
- Zero DM subtraction: Mitigation of RFI based on appearance in all channels with zero DM due to lack of dispersion.

1.2.3 Dedispersion

The radio signals are dispersed due to the interaction with the ISM. To search for FRBs this effect due to dispersion needs to be removed. For doing this the data is analyzed using many many trial DM values. Since this method is practically a brute force, in single pulse searches, this step is computationally heavy and time consuming.

1.2.4 Normalization

After dedispersion, the data is smoothened over a window, by subtracting its mean. This smoothened data is then divided using the standard deviation which provides a normalized dataset. The amplitude of each sample becomes the **Signal to Noise ratio (SN)** of the sample.

1.2.5 Matched Filtering

Once the dedispersed data is acquired, it undergoes convolution with a boxcar kernel of varying widths to search for pulses. If a specific threshold signal-to-noise (SN) value is reached, it is marked as a potential candidate and forwarded for human examination.

1.3 Current challenges in Single pulse searches

Now that a general idea of the search pipeline is established, some existing challenges in detection pipelines are as follows.

1. Due to the relatively low field of view of current FRB surveys the number of detected events is low, even though there are thousands of likely events.

2. CHIME survey with 1024 beams is continuously searching for newer candidates in the range $400 - 800MHz$ with very high DMs.[6] This increases the chances of detections.

3. But along with newer detections, the number of false positives above the threshold SN values will also increase. Hence the human inspection of all candidates will become intractable.

4. This calls for the effective use of automation in detection pipelines, instead of constant human inspections.

For automating the false positive detection process, machine learning can be effectively used .Previously effective classification using Convolutional neural networks with promising results has been done by multiple groups, including the baseline model used in this study by Connor et al. .[8]A detailed review of all the previous attempts are compiled by Agarwal et al. and can be found here[9]
This thesis focuses on comparing the pre-exisiting CNN classifier architectures trained with common stochastic gradient (SGD) methods with a much more robust Neuroevolutionary

(NE) optimisation, this method in case of FRB classification has not been explored yet to best knowledge of the author.

The rest of the thesis is structured as follows, the second chapter deals with explaining classical machine learning by giving an overview and explaining about the dataset used for training. The third section describes in detail genetic algorithms and neuroevolution. The fourth and last section discusses the comparison results as well as conclusions of the study along with remarks for future modifications.

Chapter 2

Machine learning pipelines for FRB classification

2.1 An overview of Machine learning

Machine learning (*ML*) has been one of the most widely used data analysis techniques that exists of today. The versatility of ML in performing complex computational tasks such as regression, classification, Reinforcement Learning (*RL*) etc. has proven to be extremely useful in multitude of scientific sectors including astrophysical data analysis.

ML is generally a set of tools/algorithms that allows a computer to do regression/classification tasks without explicitly defining variable relationships. These tools are commonly refereed to as 'Artificial neural networks' (*ANN*). ANNs try to replicate a neural structure, were similar to that of a biological learning neural system, by analysing the data, it tries to adjust certain parameters which eventually will 'tune' itself to do the task.

An ANN typically consists of

- Input Layer
- Hidden Layer (s)
- Output Layer

A neural network is collection 'neurons' identical to a biological neural system. These neurons are interconnected to each other, the parameters define the 'strength' of these connections are called *weights*. By optimising these weights the information transferred from one layer of neurons to the next layer is controlled.

Generally the output of a neuron can be expressed as a weighted sum of the weights and an added bias.

$$z = \mathbf{W} \cdot \mathbf{Y} + b \tag{2.1}$$

where W is the matrix of weights and Y is the input vector.

or more precisely,

$$z = \sum w_i y_i + b \tag{2.2}$$

Equation 2.3 describes the output z which is connected to that of the next neuron. This output value is then passed through an activation function. An activation function ($\mathcal{F}$) decides if the next neuron should be activated or not. This is a non linear function which maps the value into a scalar output.

$$Z = \mathcal{F}\left(\sum w_i y_i + b\right) \tag{2.3}$$

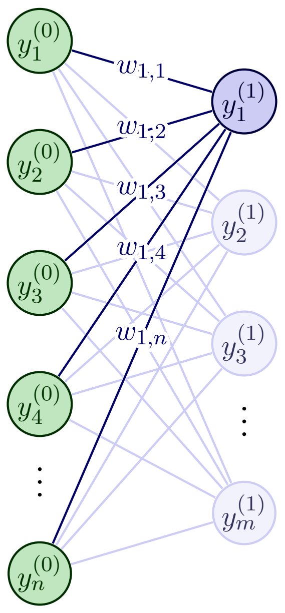
In deep learning a number of activation functions are used. Some examples include.

- Sigmoid/Logistic Function

$$\mathcal{F}(x) = \frac{e^x}{e^x + 1} \tag{2.4}$$

- Rectified linear unit (ReLU)

$$\mathcal{F}(x) = max\,(0, x) \tag{2.5}$$



$$= \mathcal{F}\left(w_{1,0}a_0^{(0)} + w_{1,1}a_1^{(0)} + \ldots + w_{1,n}a_n^{(0)} + b_1^{(0)}\right)$$

$$= \mathcal{F}\left(\sum_{i=1}^{n} w_{1,i}a_i^{(0)} + b_1^{(0)}\right)$$

$$\begin{pmatrix} z_1^{(1)} \\ z_2^{(1)} \\ \vdots \\ z_m^{(1)} \end{pmatrix} = \mathcal{F}\left[\begin{pmatrix} w_{1,0} & w_{1,1} & \ldots & w_{1,n} \\ w_{2,0} & w_{2,1} & \ldots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,0} & w_{m,1} & \ldots & w_{m,n} \end{pmatrix} \begin{pmatrix} y_1^{(0)} \\ y_2^{(0)} \\ \vdots \\ y_n^{(0)} \end{pmatrix} + \begin{pmatrix} b_1^{(0)} \\ b_2^{(0)} \\ \vdots \\ b_m^{(0)} \end{pmatrix}\right]$$

$$z^{(1)} = \mathcal{F}\left(\mathbf{W}^{(0)} y^{(0)} + \mathbf{b}^{(0)}\right)$$

Figure 2.1: An activation function acting on a neurons

- Hyperbolic tangent (tanh)

$$\mathcal{F}(x) = tanh(x) \tag{2.6}$$

A deep neural network (DNN) consists of several of these neurons stacked on top of each other as layers. It typically contains a number of hidden layers. The hidden layers in which all neurons connected to each other are called fully connected or dense layers.

2.1.1 Training of a neural network - Backpropagation

Given the learning is supervised ie, the network has access to labels to match while training. The weights and biases needs to optimized such that it performs well at the given task.To optimize the weight matrix, first requirement is some large amount data that network can be trained with, this data is termed as '*Training data*' or '*Training set*'.

First the network is initialized with a random sets of weights and biases. Now the output Z is obtained by applying the operation mentioned in Eq. 2.3. To see how much is the difference between the actual output $\mathbf{Z}$ from the input $\mathbf{Y}$ A loss function $\mathcal{F}$ can be defined, which is just the mean squared error (MSE).

$$\mathcal{F}(\mathbf{W},\mathbf{B}) = \frac{1}{N}\sum (z_i - y_i)^2 \tag{2.7}$$

where N is the number of samples, and **W** and **B** are the weights and biases corresponding to the loss function.

2.1.2 Minimizing the loss function

To minimize the loss and further optimize the weights, different strategies are employed. Two of them are discussed below.

Stochastic Gradient Descent

This method is generally used for minimising the loss function. To see this, first the loss function mentioned in Eq.2.7 is expanded in taylor series to the first order.

$$\mathcal{F}(\mathbf{W}+\Delta\mathbf{W},\mathbf{B}+\Delta\mathbf{B}) \approx \mathcal{F} + \frac{\partial\mathcal{F}}{\partial\mathbf{W}}(\Delta\mathbf{W}) + \frac{\partial\mathcal{F}}{\partial\mathbf{B}}(\Delta\mathbf{B}) = \mathcal{F} + \nabla^T\mathcal{F}\Delta\mathbf{W}\Delta\mathbf{B} \tag{2.8}$$

To minimize the loss function, the term $\nabla^T\mathcal{F}\Delta\mathbf{W}\Delta\mathbf{B}$ as negative as possible. This parameter is updated accordingly, by defining a training parameter η such that ;

$$\kappa \leftarrow \kappa - \eta\mathcal{F}(\kappa) \tag{2.9}$$

Where $\kappa = \Delta\mathbf{W}\Delta\mathbf{B}$

The SGD method uses small steps iteratively for convergence. This method is generally termed as *Backpropagation*. SGD is regularly termed as an 'optimizer', since it optmises the parameter matrix. More generally, the values of the number of neurons, the weights, activation function which constitute a neural networks are called *Hyperparameters*. The tuning of these hyperparameters involves various techniques which is beyond the scope of this thesis.

Evolutionary Optimisation

In the previous case, the optimisation is done by taking small steps towards convergence by minimising the loss function. This method works very well, provided that the task at hand has a 'local' solution. But, this algorithm is prone to get stuck on a local minima. This prevents further training of the network.
An alternative to this is to use a global optimisation method. Optimisations based on evolutionary tactics, which are directly inspired from biological evolution has proven to be better in those scenarios. A detailed discussion of this will be done in Chapter 3.

2.2 Convolutional Neural networks

Convolutional neural networks (CNNs), are a class of neural networks which are specifically designed to train using image data. CNNs are similar to the working of an ordinary feedforward network, but to extract features of an image, additional layers are also included.[10].
A CNN, as the name suggests includes a convolutional layer. This layer is specifically used to perform convolutional operations on the input data matrix. This layer consists of kernels, which convolves with the matrix, to extract features out of it. The kernel weights are similarly trained as a conventional neural net. After the convolution is performed, the image array is reduced down to a smaller dimension, by performing pooling. Pooling is done either by averaging the global image values (Global average pooling) or by taking a specific window size (Max average pooling).[11]

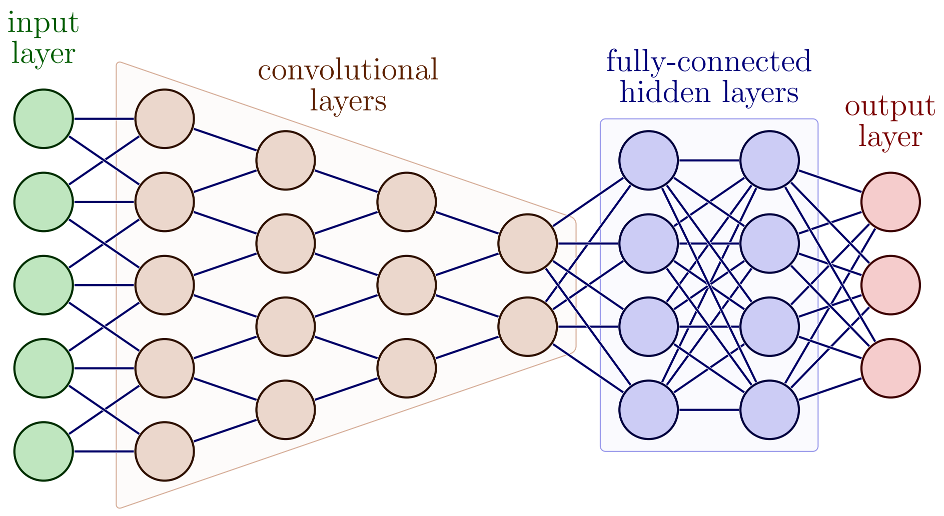


Figure 2.2: A CNN architecture

After this the values are flattened and directly fed into a regular dense layer which is further directed to an output layer.

2.3 FRB classification using CNNs

CNNs are efficient in extracting features from image data. As mentioned in chapter 1, we can construct datasets which represent a FRB. This includes,

- *Frequency - Time (Waterfall) data*
- *DM - Time data*
- *De-dispersed Pulse profiles*

For this analysis, the above mentioned datatypes are utilized.

2.3.1 Datasets

To perform the classification, a valid dataset need to be used. The training images needs to be reshaped to a uniform dimension. Data obtained from various surveys (CHIME, Parkes, etc.) needs to be reshaped to a uniform dimension first before being fed into the network.

Instead of doing this, FRBs can be synthesised based on some parameters. This will only include single pulse profiles, but by varying the synthesising parameters, FRBs with different SNRs can be obtained. Here, for this work, an openly available FRB simulation framework 'Single-pulse-ml' is utilized developed by Liam Connor [1].

. This framework generates 10000 FRBs candidates out of which 5000 are false positives. The snrs are taken from a log normal distribution, the snr distribution used is shown below.

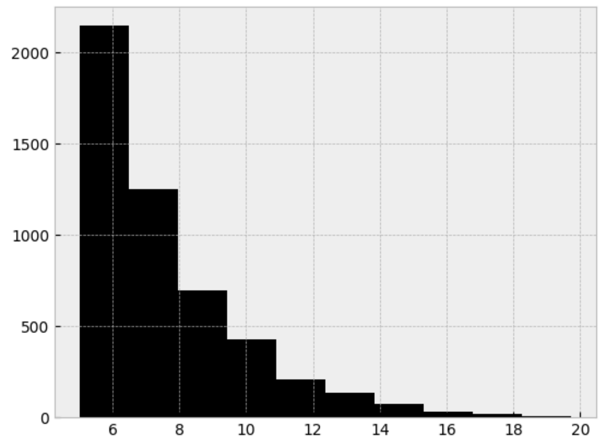


Figure 2.3: SNR distribution of synthesised data.

Obtained data

A single pulse candidate of the synthesised data is shown below. Along with a false positive from the same dataset. The DM-time data appears to be the characteristic 'bowtie' pattern. This dataset has mean SN ratio of approximately 8.8. any other higher SN single pulse can effectively added to the dataset for training it, since the network already is equipped to classify lower SN false positives.

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

- FRB candidate , SNR = 6.22

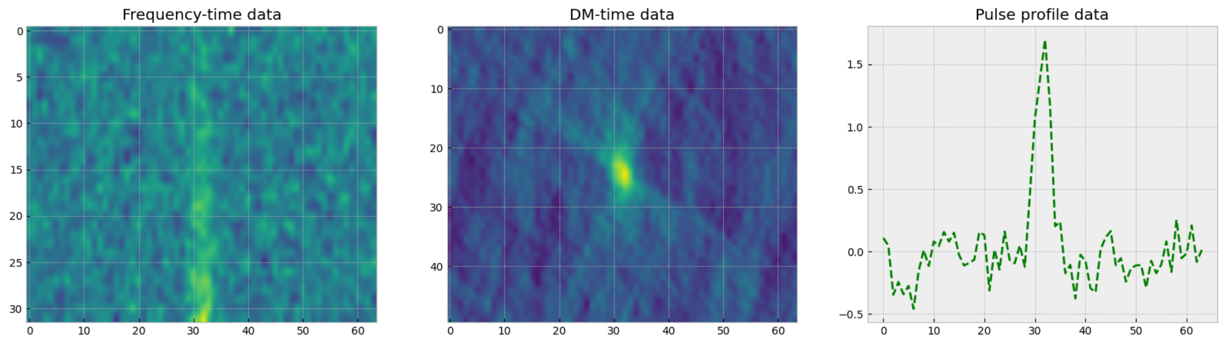


Figure 2.4: FRB true candidate, Frequency-time,DM-time,Pulse profile plots respectively.

- RFI candidate , SNR = 5.95

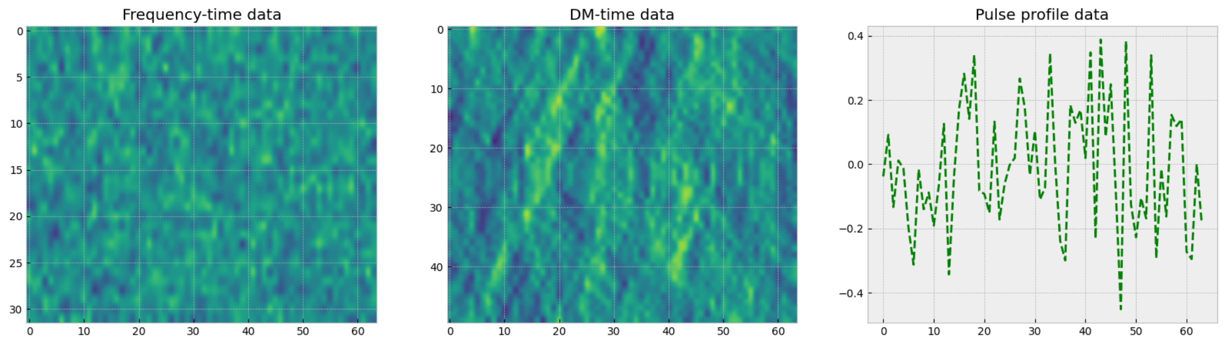


Figure 2.5: FRB false positive candidate, Frequency-time,DM-time,Pulse profile plots respectively.

While training a neural network, a bias in the classification data, can severely impact the performance of the results. To counter this, the dataset is then thoroughly shuffled. Equal number of true and false positives are selected to ensure there is no training bias.

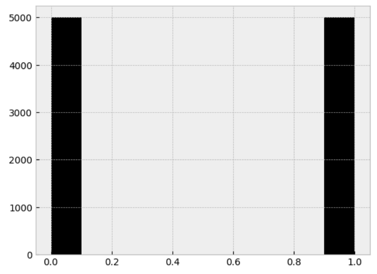


Figure 2.6: Distribution of candidates after shuffling

Chapter 3

Genetic Algorithms and Neuroevolution

3.1 Introduction and implementation of Neuroevolutionary (NE) algorithms

As mentioned in subsection 2.1.2, different optimisation backends can be implemented. The major focus of this thesis is to explore and compare between traditional gradient descent methods and evolutionary methods. Genetic evolution is one of the most used evolutionary algorithms, in the class of multiple other types, ie, Particle swarm optimisation (PSO), Differential evolution (DE), Ant colony optimisation (ACO), etc. A discussion on these algorithms is beyond the scope of this thesis, a detailed overview can be found here. [12]

3.1.1 Genetic algorithms

Genetic algorithm is a meta heuristic algorithm inspired from the principle of Darwinian evolution.[13] Nature selects the most fittest individual by testing the adaptability of an individual organism to the environment itself. The algorithm was theorized by John Holland.[14] as an optimization method in the early 1980s. The algorithm tries to find the

best possible solution from a number of solutions provided. The evolution is based on the idea of creation of parents, their traits are crossed over creating a daughter population, then the best suited individual from this pool is selected.
GAs have been used in multiple optimization problems in radio astrophysics as well. GAs have been widely used in antenna design optimisations.[15]

Genetic operators

After the parent pool is generated for the solution of a task, different genetic operations are imposed on it through an iterative process. These genetic operations are given below.[16]

- Selection
- Crossover
- Mutation

These basic operations are performed in every generation of the population created.

1. **Selection:** Selection of a parent is dependant on the '*Fitness*' of the parent. A fitness is defined as a numerical score that is given for that solution for completing the task. Fitness are defined accordingly to the task at hand. A higher fitness score is always preferred for the selection of an individual.

2. **Crossover:** After certain number of parents are selected, the parents are allowed to '*exchange*' favourable features that they have. This directly corresponds to the crossover that is undergone in biological species. This ensures every generation produces better individual with a new genetic code.

3. **Mutation:** Even though crossover produces new individuals with better architecture, this doesn't necessarily can lead to a converging solution, often the daughter pool can get stuck with similar type of solutions, to prevent this, and ensure genetic variety, random mutations in the genetic code are introduced. This produces new exclusively new type of individuals, and ensures that solutions doesn't diverge.

Steps $1 - 3$ are repeated until a convergent solution is obtained. The entire algorithm is given below in flowchart below.

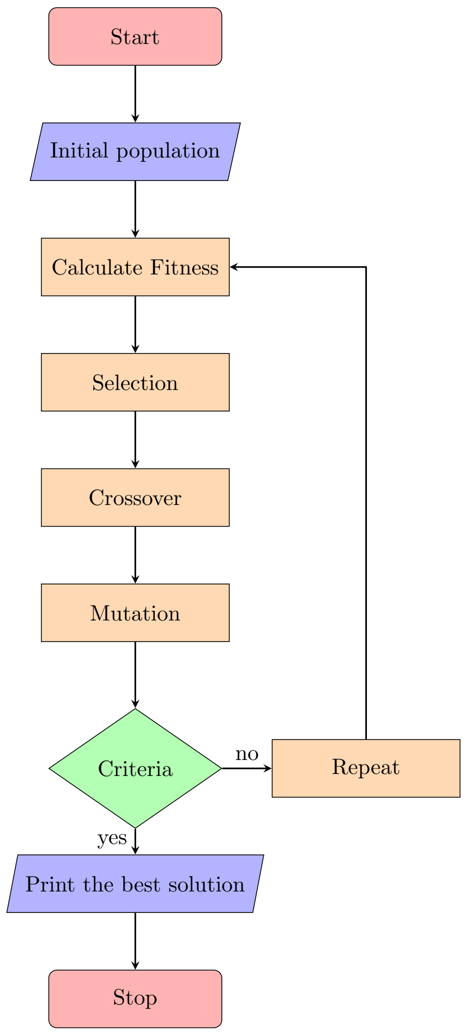


Figure 3.1: Genetic Algorithm - Flowchart

After the convergence has been obtained from the iterative process, the algorithm stops by outputting the best solution.

Parent selection

Each run of the genetic instance is called a generation. After each generation a parent is selected for the next instance. It is possible that the one best fit individual dominates newer generations. This prevents a diverse population pool, which is crucial for the convergence of a GA.

To prevent this multiple selection procedures have been proposed, some of them are listed below.[16]

- **Roulette Wheel Selection/Fitness proportional Selection: (RWS)** This selection is based on selecting an individual by summing all their fitness scores and then dividing each individual fitness with this sum. A random number between chosen between 0 and 1. According to this the parent is selected.
- **Tournament Selection: (TS)** In this selection a number (*Tournament size*) of individuals (*Tournament*) are selected and the winners (*highest fitness score*) are selected from this subset.
- **Stochastic Universal Selection (SUS):** Similar to that of RWS, but instead of one selection point, there are multiple selection points and the required number of parents are selected at once, instead of spinning the roulette wheel multiple times.

Crossover operations

Crossover operation is extremely important as it creates the next generation of the process. Crossover splits to parents and recombines them into a daughter individual.This operation is highly dependant on the representation of the individual used. A few of commonly used crossover operations are listed below.

- **One point crossover:** The most simplest approach in crossover is to split the genomes of the corresponding parents are combine them.
- **Two point crossover:** In contrast to one point crossover, this method separates the genome in two different points. Thus reducing bias in splitting.
- **Blend Crossover:** The input genomes are 'blended' together by taking their arithmetic mean to create the daughter vector. Used in case of real number representations.

Mutation Operations

Mutation operation can also be done in multiple ways.

- **Bit flip mutation:** Commonly used in binary representations, a ’bit’ (0 or 1) is randomly selected and flipped.
- **Delta mutation :** This type of mutation is generally used in real number representations. An real number Δ is incremented or decremented accordingly from individual.
- **Gaussian mutation:** Similar to delta mutation, the Δ value is drawn from a gaussian distribution.

3.1.2 Neuroevolution - Generating better neural networks

The algorithm mentioned in the previous section is extremely efficient and is better adapted to noise due to its structure[17]. A striking feature of GAs are they can be implemented to optimize any type of population. This is where the idea of neuroevolution originally theorized.

Neuroevolution uses GAs as a backend to generate populations of neural networks that can be specifically designed for a task.

Evolution to ANNs have been introduced in multiple levels. Since the GAs have been so versatile on learning using different types of entry types (genotypes), provided that they can be undergo the genetic operations mentioned in subsection 3.1.1. These features can be introduced in :

- Connected weights and biases
- Overall architecture (NEAT).
- Learning Rules.

Encoding the information for the GA to process also becomes significant in this case, Similar to what mentioned in subsection 3.1.1.

Optimising connected weights and biases

This thesis tries to explore the simplest kind of neuroevolution, where the connected weights and biases are optimized. Before being fed into the genetic algorithm, the neural network needs to be encoded. A naive way of doing this is to encode the network as an adjacency matrix.[16] Consider a classic layered network with input layer with 2 neurons, one hidden layer with 3 neurons and an output later with one neuron. For simplicity the weights are assumed to binary, where connection between two neurons means a weight of 1 otherwise, 0. The adjacency matrix of this fully connected network is given below.

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

This adjacency matrix can be plotted as a network, as shown below.

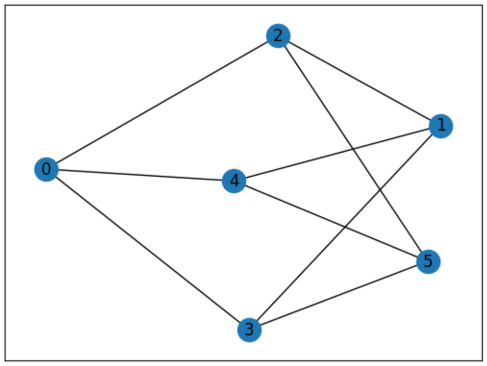


Figure 3.2: An example graph of a fully connected network given in the example matrix.

To illustrate the neuroevolution process in more detail, consider a simple example. Let A and B be two parent neural networks with identical structures but with different sets of binary weights. ie,

$$A = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} ; B = \begin{bmatrix} 0 & 0 & 2 & 2 & 2 & 0 \\ 0 & 0 & 2 & 2 & 2 & 0 \\ 2 & 2 & 0 & 0 & 0 & 2 \\ 2 & 2 & 0 & 0 & 0 & 2 \\ 2 & 2 & 0 & 0 & 0 & 2 \\ 0 & 0 & 2 & 2 & 2 & 0 \end{bmatrix}$$

According to subsection 3.1.1, These parent networks should now undergo a crossover. In this case, we assume a random crossover.

$$RandCross(A, B) = C \implies \begin{bmatrix} 0 & 0 & 2 & 2 & 2 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 2 & 2 & 0 & 0 & 0 & 2 \\ 0 & 0 & 2 & 2 & 2 & 0 \end{bmatrix}$$

Introducing mutation also yeilds,

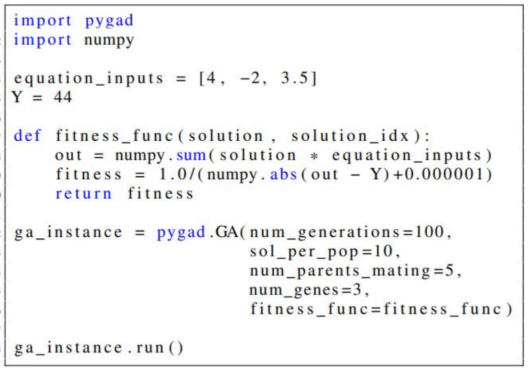
$$Mutate(C) = D \implies \begin{bmatrix} 0 & 0 & 2 & 2 & 3 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 2 & 2 & 0 & 0 & 0 & 1 \\ 2 & 2 & 0 & 0 & 0 & 1 \\ 2 & 2 & 0 & 0 & 0 & 2 \\ 0 & 0 & 2 & 2 & 2 & 0 \end{bmatrix}$$

Now matrix D will be the final daughter matrix, and the fitness of this network representation is evaluated. And the process continues till a convergent solution is obtained.

PyGAD - A viable alternative for complex encoding

The example stated previously involved a very trivial linear feedforward network, which is extremely easy to implement. But in this case, where the network has multiple number of input neurons, multiple hidden layers etc, encoding them manually becomes impossible. To get around this problem and to test multiple networks, a recently developed genetic algorithm backend for python 'PyGAD[18]', was put into use. To use PyGAD effectively, first the structure of a parent network was chosen, then it was trained using synthesized FRB data (Section 2.3.1.Then, the network is loaded as a Keras[19] functional model into the PyGAD framework. It was allowed to train for multiple number of generations and accordingly as the fitness score of each daughter network steadily increased.

A feature of PyGAD is that it allows to have custom fitness functions specific to the problem. The fitness function in PyGAD can be defined as the code snippet given below.



```
import pygad
import numpy

equation_inputs = [4, -2, 3.5]
Y = 44

def fitness_func(solution , solution_idx):
    out = numpy.sum(solution * equation_inputs)
    fitness = 1.0/(numpy.abs(out - Y)+0.000001)
    return fitness

ga_instance = pygad.GA(num_generations=100,
                       sol_per_pop=10,
                       num_parents_mating=5,
                       num_genes=3,
                       fitness_func=fitness_func)

ga_instance.run()
```

Figure 3.3: Definition of fitness function in PyGAD framework [18]

In the next section all the results obtained by studying the performance of both SGD and GA based training are presented along with conclusions and remarks.

Chapter 4

Results

4.1 Comparison of metrics

4.1.1 Simple CNN architecture

The evolutionary backend was set up using the PyGAD[18] framework. The algorithm was based on the simplest neuroevolution; where the weights and biases are updated accordingly using a genetic backend. All the related python code used for obtaining these results can be found in the github repository[1]

The following CNN was designed for the Genetic algorithm to train. The architecture, is chosen to have a simplistic layout, which is structured as follows,

Table 4.1: Simple CNN model

Layer (Type)	**Output Shape**	**Activation**
InputLayer	(32, 64, 1)	-
Conv2D	(30, 62, 5)	ReLU
MaxPooling2D	(6, 12, 5)	-
Conv2D	(4, 10, 3)	ReLU
Flatten	(120,)	-
Dense	(15,)	ReLU
Dense	(1,)	Sigmoid

This CNN for testing, is tested with both regular (SGD) based optimisation techniques

[1] https://github.com/cup-cake-lover

as well as using a genetic algorithm. To evaluate the training process in case of the GA algorithm, fitness v/s number of generations are plotted. The fitness score as mentioned in section 3.1.1, was defined as follows,

$$Fitness = \frac{1}{Loss + \delta} \tag{4.1}$$

Where δ is a very small value (≈ 0.000001) to prevent the fitness from blowing up.

For each architecture trained using SGD and GA backends, 3 evaluation metrics are chosen. These evaluation metrics are based on the parameters obtained from the confusion matrix, defined as follows.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.2}$$

$$Precision = \frac{TP}{TP + FP} \tag{4.3}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.4}$$

Where TP,TN,FP,FN are true positives, true negatives, false positives and false negatives respectively.

SGD as backend

For training using regular backpropagation method, the optimizer was set to be Adam and the network was trained for 100 epochs keeping the batch size of 50.

- **<u>Training Accuracy and loss</u>**

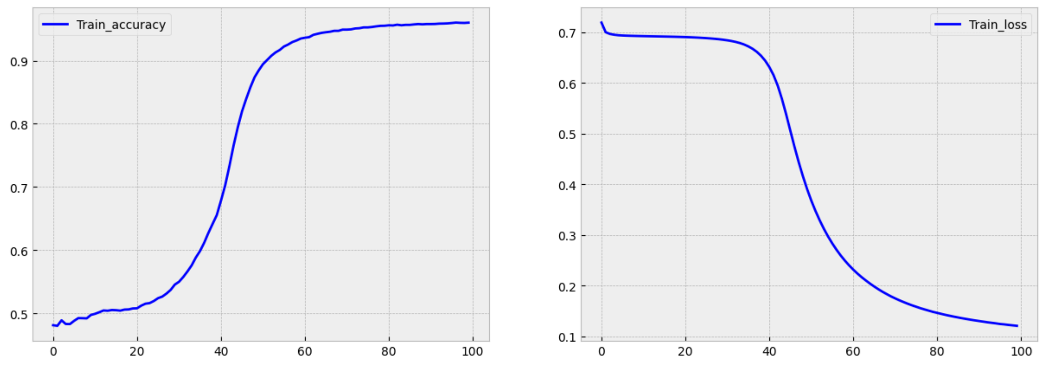


Figure 4.1: Accuracy Loss v/s Epochs

- **<u>Validation Accuracy and loss</u>**

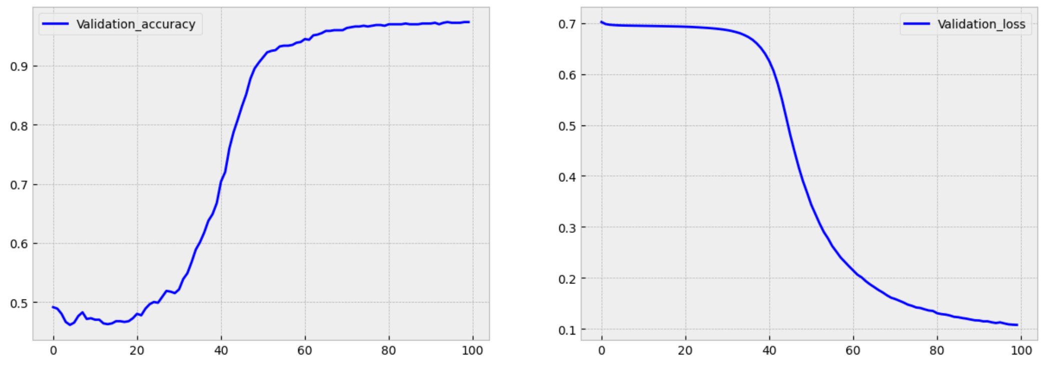
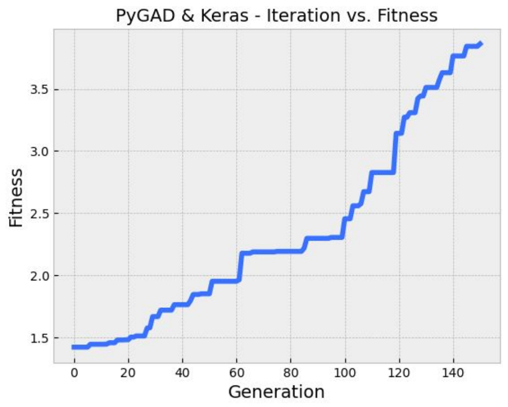


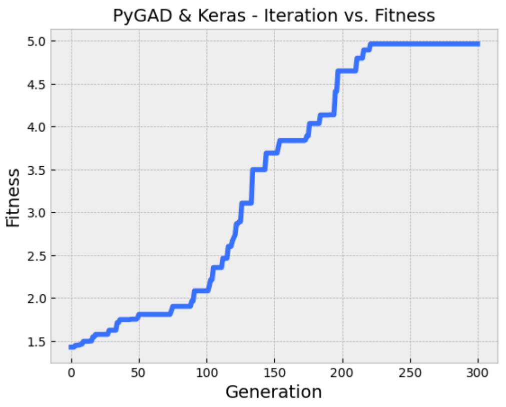
Figure 4.2: Accuracy Loss v/s Epochs

Genetic Algorithm as backend

Figure 4.3: 2D simple CNN - Frequency time data - Fitness v/s Generations



(a) Fitness v/s Generations - 250 gen



(b) Fitness v/s Generations - 100 gen

Table 4.2: Evaluation metrics for simple CNN - both SGD and GA backends.

Metric	**SGD CNN**	**GA CNN - 300 generations**
Accuracy	96.35%	91.80%
Precision	97.79%	97.23%
Recall	94.71%	85.77%

4.1.2 CNN model - version 2

SGD backend

To compare the performance of the GA based training, a previously benchmarked CNN is then used. The Following CNN is based on the architecture given by [8] The architecture is implemented using Tensorflow keras[20]. The architecture used is as follows.

Table 4.3: 2D CNN models (*For Frequency-time and DM-time data*)

Layer (Type)	**Output Shape**	**Activation**
InputLayer	(32, 64, 1)	-
Conv2D (Kernel Size = 3x3,strides = 2)	(30, 62, 5)	ReLU
MaxPooling2D	(15, 31, 5)	-
Dropout	(0.5)	-
Conv2D (Kernel Size = 5x5,strides = 2)	(13, 29, 3)	ReLU
MaxPooling2D	(6, 14, 3)	-
Flatten	(252,)	-
Dropout	(0.6)	-
Dense	(1,)	Sigmoid

For frequecy-time and DM-time, similar architectures is kept. For dedispersed pulse profiles, a similar $1D$ convolutional network was used.

Table 4.4: 1D CNN Architecture (*For Pulse profiles)*

Layer (Type)	**Output Shape**	**Activation**
InputLayer	(64, 1)	-
Conv1D	(30, 32)	ReLU
MaxPooling1D	(15, 32)	-
Conv1D	(6, 64)	ReLU
Flatten	(384,)	-
Dense	(1,)	Sigmoid

The training data was split into 80% training 20% testing. While training, 1% split was made in the initial training split for validation testing. The models was trained for 100 epochs with a batch size of 50.

- **<u>Frequency - Time data.</u>**

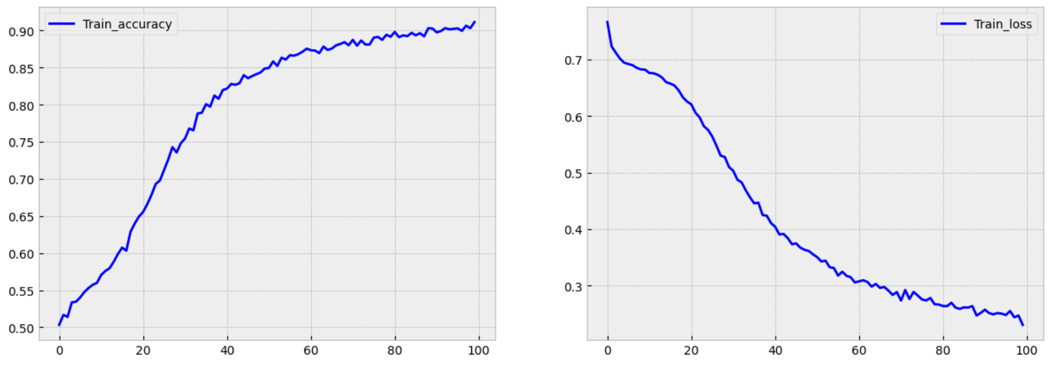


Figure 4.4: Training metrics of Frequency-time data - Accuracy and Loss

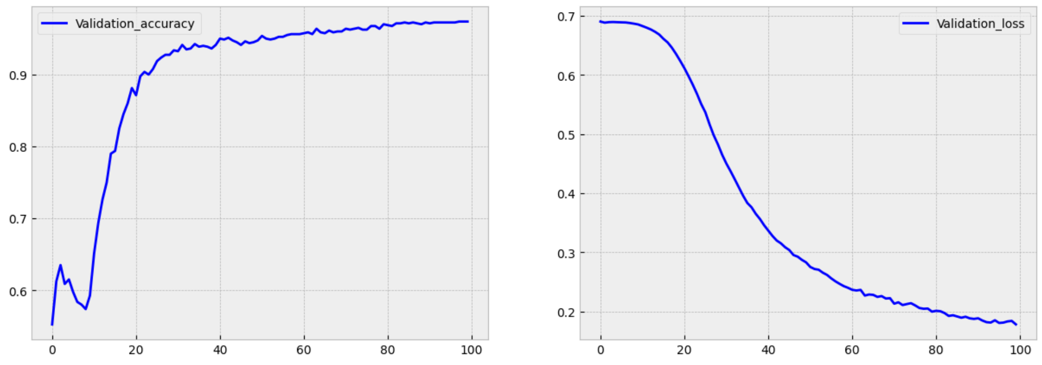


Figure 4.5: Training metrics of Frequency-time data - Validation Accuracy and Loss

- **<u>DM - Time data.</u>**

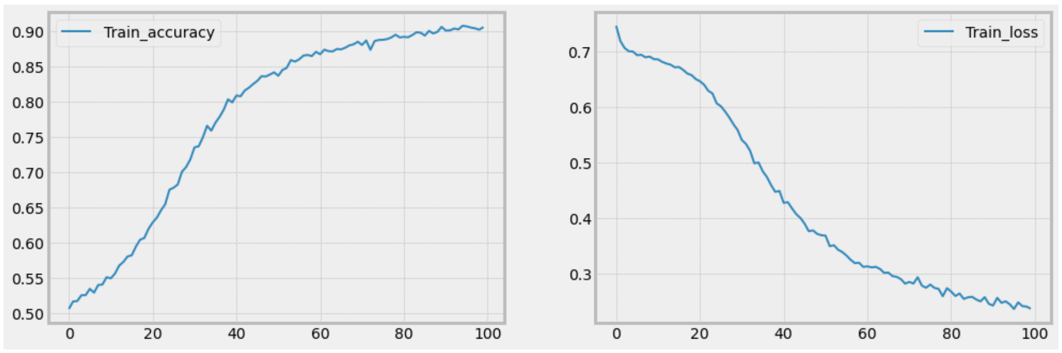


Figure 4.6: Training metrics of DM-time data - Accuracy and Loss

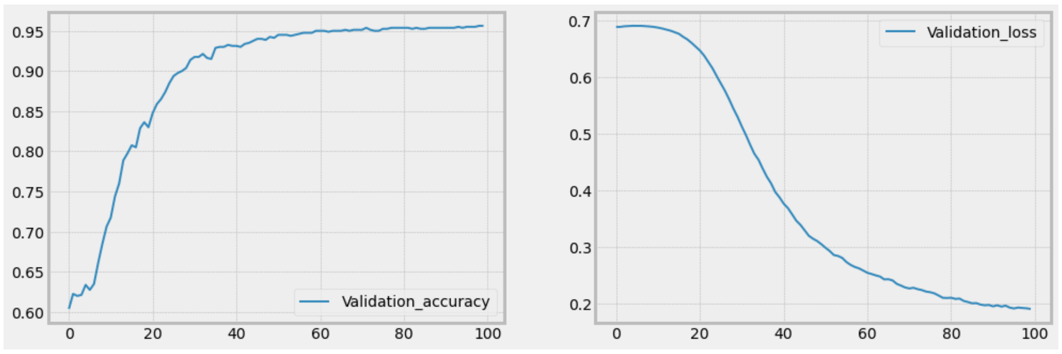


Figure 4.7: Training metrics of DM-time data - Validation Accuracy and Loss

- **<u>Dedispersed pulse profile data.</u>**

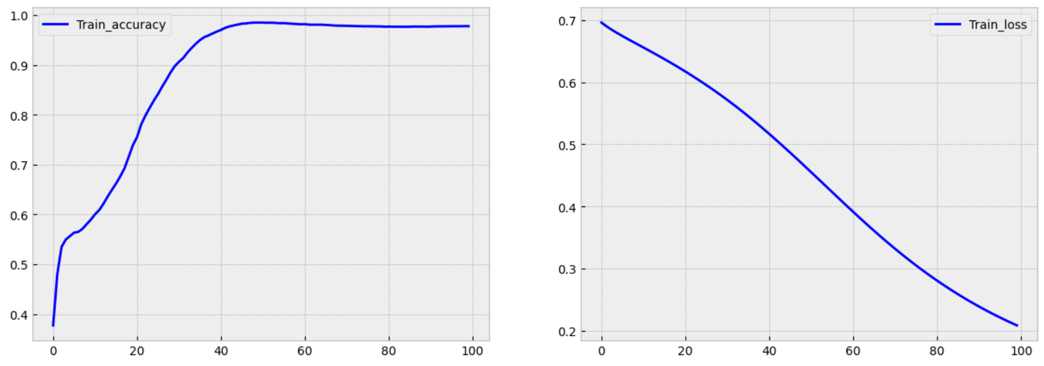


Figure 4.8: Training metrics of Pulse profile data - Accuracy and Loss

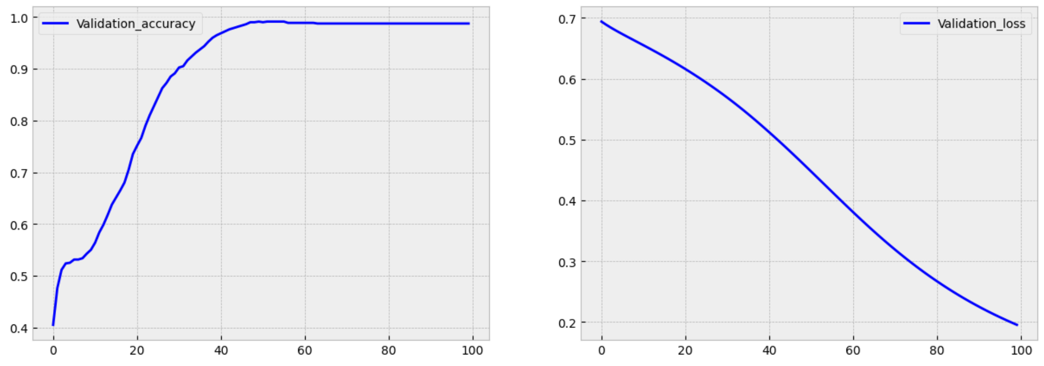


Figure 4.9: Training metrics of Pulse profile data - Validation Accuracy and Loss

Table 4.5: Evaluation Metrics - Modified CNN - SGD

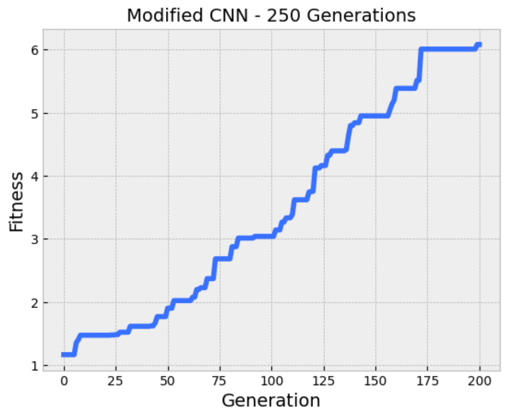
Metric	**Frequency-time model**	**DM-time model**	**1D pulse profiles**
Accuracy	96.85%	97.98%	98.20%
Precision	96.94%	99.11%	99.89%
Recall	96.94%	94.49%	96.44%

Genetic Algorithm backend

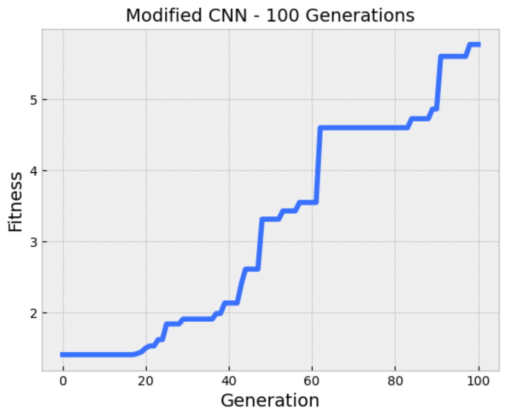
In this case, the genetic algorithm was allowed to run with the modified CNN as its seed for both 250 and 100 generations. In case of 250 generations, the number of parents that were mated was set to 5, while in case of the 100 generation run, the number of parents were reduced to two to observe possible differences.

1. 2D CNN - Frequency-time data

Figure 4.10: 2D CNN - Frequency-time data - Fitness v/s Generations



(a) Fitness v/s Generations - 250 gen



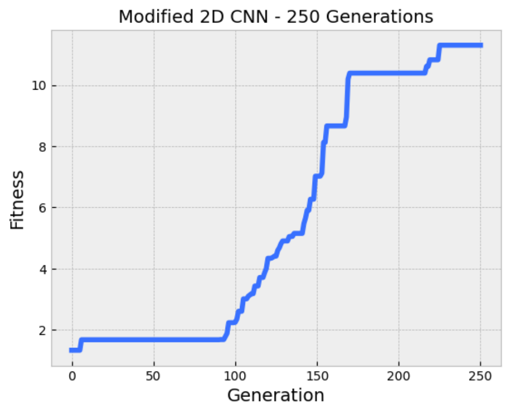
(b) Fitness v/s Generations - 100 gen

Table 4.6: Evaluation Metrics for modified GA-CNN - Frequency-time Dataset

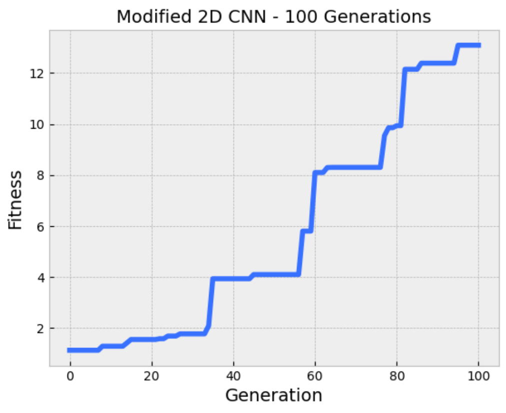
Metric	**250 Generations**	**100 Generations**
Accuracy	95.60%	95.80%
Precision	98.06	96.48%
Recall	92.88	94.91%

2. 2D CNN - DM-time data

Figure 4.11: 2D CNN - DM-time data - Fitness v/s Generations



(a) fitness v/s Generations - 250 gen



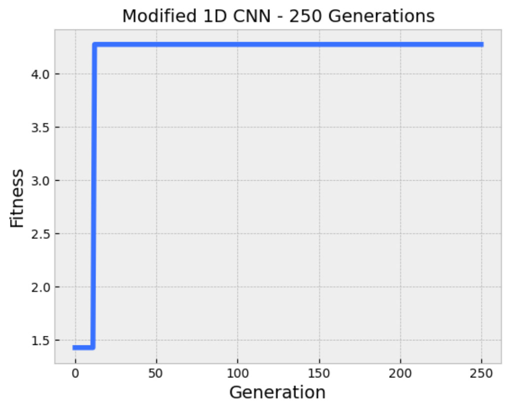
(b) Fitness v/s Generations - 100 gen

Table 4.7: Evaluation Metrics for modified GA-CNN - DM-time dataset

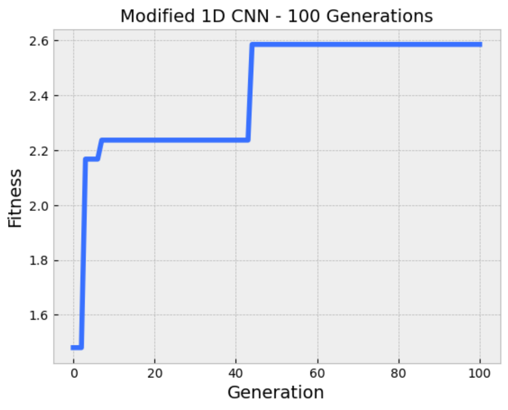
Metric	**250 Generations**	**100 Generations**
Accuracy	99.25%	98.85%
Precision	99.48%	99.79%
Recall	98.98%	97.86%

3. 1D CNN - Pulse Profile data

Figure 4.12: 2D CNN - Pulse prfile data - Fitness v/s Generations



(a) Fitness v/s Generations - 250 gen



(b) Fitness v/s Generations - 100 gen

Table 4.8: Evaluation Metrics for modified GA-CNN - Pulse profile dataset

Metric	**250 Generations**	**100 Generations**
Accuracy	91.05%	93.00%
Precision	98.90%	94.42%
Recall	82.72%	91.15%

4.2 Discussion of results and Conclusions

From the results obtained from both SGD and GA optimisers, the accuracy values obtained were comparable and high for both cases. The most striking difference observed was the training process of both backends. Neuroevolution in this case was slower and was often stuck on a fitness score for a significant number of generations. But eventually due to randomness introduced by the mutation operators, The fitness score jumps. This is evidently seen in the fitness v/s generations plot.

Other factor that became crucial was the datatype used. Since the three types of datasets were used (Frequency - time, DM-time and Pulse profiles.) DM-time data gave significantly better results ($\approx$ 98% accuracy). Hence one major change that is required for a seed architecture is the ability to use all three data types simultaneously. This will greatly improve the efficiency of the network architecture.

The GA based training was done for two sets of generations (250 and 100) with different number of parents (5 and 2 respectively). In case of all the training instances, the set with 2 parents that ran for 100 generations gave much better results. This was unexpected, since number of generations was supposed to constrain the fitness scores. But performed much better and faster.

In this particular test case, although stochastic gradient descent demonstrated comparable performance to the Genetic backend, there are some notable constraints associated with using the Genetic Algorithm (GA). These constraints include:

- Very simple neuroevolution scheme (Based on adjacency matrices)
- Constant mutation rates. (This limits the degree of change in the hyperparameters)

- Non-parallelized execution.

Neuroevolution is expected to perform better if the following changes can be made.

1. Instead of just optimising weights, entire architecture can be augmented while the training is done (NEAT)[21]. This completely ensures production of newer and more robust networks.

2. In the context of Fast Radio Burst (FRB) classification, the continuous expansion of the FRB database and the detection of new signal profiles necessitate constant modification of the classification network. Traditional convolutional neural networks (CNNs) often require the development of entirely new architectures to accommodate these changes. However, NEAT (NeuroEvolution of Augmenting Topologies) ensures that a classifier tailored to the specified requirements is always generated.

   By using NEAT, the classification network is continually refined and updated, enabling it to handle the ever-expanding FRB database effectively. The evolutionary nature of NEAT allows for the exploration of different network configurations and the selection of the most suitable ones based on performance and specification requirements. This adaptability ensures that the classifier remains up-to-date and capable of accurately classifying new FRB signals as they are detected.

3. An another possibility is the ability of the network to optimize learning rules. This will also increase the classification accuracy[22].

4. One big advantage of NE, which was not explored here, is parellelisation. The entire process of generation, fitness calculation mutation etc can be completely parllelized[23]. This is extremely crucial when such a framework is integrated with existing search pipelines. The gives rise to the opportunity of generating a hundreds of parent networks and then their evaluation, which happens simultaneously. Which significantly increases the efficiency of search pipelines.

In summary, the current research focuses on a preliminary exploration of neuroevolution and its potential application in FRB classification. The purpose is to establish a foundational framework that can serve as a starting point for further enhancements, incorporating the aforementioned remarks. By building upon this initial work, more advanced techniques can be developed, offering a competitive alternative to existing frameworks such as Generative Adversarial Networks (GANs)[24] in the field of FRB classification.

Bibliography

[1] A. HEWISH et al. "Observation of a Rapidly Pulsating Radio Source". In: *Nature* 217 (Feb. 1968), pp. 709–713. DOI: 10.1038/217709a0.

[2] Richard N Manchester et al. "The Parkes multi-beam pulsar survey - I. Observing and data analysis systems, discovery and timing of 100 pulsars". In: *Monthly Notices of the Royal Astronomical Society* 328 (Nov. 2001), pp. 17–35. DOI: 10.1046/j.1365-8711.2001.04751.x.

[3] D. R. Lorimer et al. "A Bright Millisecond Radio Burst of Extragalactic Origin". In: *Science* 318 (Nov. 2007), pp. 777–780. DOI: 10.1126/science.1147532.

[4] D. Thornton et al. "A Population of Fast Radio Bursts at Cosmological Distances". In: *Science* 341 (July 2013), pp. 53–56. DOI: 10.1126/science.1236789.

[5] Emily Petroff et al. "VOEvent standard for fast radio bursts". In: *arXiv e-prints* (Oct. 2017), arXiv:1710.08155. DOI: 10.48550/arXiv.1710.08155.

[6] Mandana Amiri et al. "The First CHIME/FRB Fast Radio Burst Catalog". In: *The Astrophysical Journal Supplement Series* 257 (Dec. 2021), p. 59. DOI: 10.3847/1538-4365/ac33ab.

[7] D. Anish Roshi et al. "Advanced multi-beam spectrometer for the Green Bank Telescope". In: (Oct. 2011). DOI: 10.1109/ursigass.2011.6051280.

[8] Liam Connor and Joeri van Leeuwen. "Applying Deep Learning to Fast Radio Burst Classification". In: *The Astronomical Journal* 156 (Nov. 2018), p. 256. DOI: 10.3847/1538-3881/aae649.

[9] Devansh Agarwal. "Searches for Fast Radio Bursts using Machine Learning". In: (Jan. 2020). DOI: 10.33915/etd.7827.

[10] Skikri Driss et al. *A comparison study between MLP and Convolutional Neural Network models for character recognition.* May 2017, p. 1022306. DOI: 10.1117/12.2262589.

[11] Florentin Bieder, Robin Sandkuehler, and Philippe Cattin. *Comparison of methods generalizing max- and average-pooling.* Mar. 2021.

[12] Farid Ghareh Mohammadi, Amini M Hadi, and Hamid R Arabnia. "Evolutionary Computation, Optimization and Learning Algorithms for Data Science". In: *arXiv (Cornell University)* (Aug. 2019). DOI: 10.48550/arxiv.1908.08006.

[13] R. Balamurugan, A. M. Natarajan, and K. Premalatha. “Stellar-Mass Black Hole Optimization for Biclustering Microarray Gene Expression Data”. In: *Applied Artificial Intelligence* 29 (Apr. 2015), pp. 353–381. DOI: 10.1080/08839514.2015.1016391.

[14] John H Holland. “Genetic algorithms”. In: *Scientific American* 267.1 (1992), pp. 66–73.

[15] E.E. Altshuler and D.S. Linden. “Wire-antenna designs using genetic algorithms”. In: *IEEE Antennas and Propagation Magazine* 39 (Apr. 1997), pp. 33–43. DOI: 10.1109/74.584498.

[16] Ka-Chun Wong. “Evolutionary algorithms: Concepts, designs, and applications in bioinformatics: Evolutionary algorithms for bioinformatics”. In: *CoRR* abs/1508.00468 (2015).

[17] T Then and Edwin Chong. “Genetic Algorithms in Noisy Environments”. In: *ECE Technical Reports Electrical and Computer Engineering* 12 (1993).

[18] Ahmed Fawzy Gad. *PyGAD: An intuitive genetic algorithm python library*. 2021.

[19] François Chollet. *Keras*. 2015.

[20] Martín Abadi et al. *TensorFlow: Large-scale machine learning on heterogeneous systems*. Tensorflow, 2015.

[21] Kenneth O. Stanley and Risto Miikkulainen. “Evolving Neural Networks through Augmenting Topologies”. In: *Evolutionary Computation* 10 (June 2002), pp. 99–127. DOI: 10.1162/106365602320169811.

[22] Kenneth O Stanley. “Compositional pattern producing networks: A novel abstraction of development”. In: *Genetic Programming and Evolvable Machines* 8 (2007), pp. 131–162.

[23] John Runwei Cheng and Mitsuo Gen. “Parallel Genetic Algorithms with GPU Computing”. In: *Industry 4.0 - Impact on Intelligent Logistics and Manufacturing* (Mar. 2020). DOI: 10.5772/intechopen.89152.

[24] Ian J Goodfellow et al. “Generative Adversarial Networks”. In: *arXiv (Cornell University)* (June 2014). DOI: 10.48550/arxiv.1406.2661.

[25] E Petroff, and Duncan R Lorimer. “Fast radio bursts at the dawn of the 2020s”. In: *The Astronomy and Astrophysics Review* 30 (Mar. 2022). DOI: 10.1007/s00159-022-00139-w.

[26] Xin Yao. “Evolving artificial neural networks”. In: *Proceedings of the IEEE* 87 (1999), pp. 1423–1447. DOI: 10.1109/5.784219.