Comparison and Evaluation of Neural Network Architectures

Author
Akarshan Jaiswal

School of Mathematics and Computer Science

A thesis presented for the degree of
MSc. Data Science

Supervisor
Chengjia Wang

Heriot-Watt University
April 17, 2024

Number Pages: 71

Declaration of Authorship

I, Akarshan Jaiswal, declare that this thesis titled, ‘Comparison and Evaluation of Neural Network Architectures’ and the work presented in it is my own. I confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: Akarshan Jaiswal

Dated: April 17, 2024

Acknowledgement

I want to thank my family and all the close friends and people for the encouragement throughout the studies. Apart from them I would like to give special thank my current supervisor **Dr. Chengjia Wang** and my personal tutor **Dr. Arash Eshghi**, as without their guidance and support the project would not have been possible. Special acknowledgement also goes to my previous supervisor **Dr. Benjamin Kenwright** who helped me in formulating my thesis proposal and setting this idea for the project.

Comparison and Evaluation of Neural Network Architectures

Akarshan Jaiswal

Abstract

This projects aims at implementation and training of different types and orientation of Neural Network architectures with respect to different Mathematical equations and problems. Following a comparative study on how different architectures react with different sets of Mathematical equations/problems.
These Mathematical problems range from a simple straight forward equation
(e.g. $[a + b = c]$) to a bit complex equations (e.g. $[\int a^2 cos(a) = b]$) in nature too.

As Neural networks play an extremely important role in today's landscape of AI and Data Science tools. Hence, understanding the underlying capability of these Algorithms under different architectures becomes extremely important. The following thesis aims at exposing the underlying mathematical equations that drive these different architectures, including Perceptrons, Feed-Forward neural networks, Multi-Layer Feed-Forward neural networks, Convolutional Neural Networks (CNN) and many more.

Additionally, this thesis explores the impact of different architectures on the Neural networks ability to recognize/capture complex patterns and generalize problems effectively. This also investigates the adaptability of the following network architectures towards various mathematical problems, while shining light on their fitness for different applications.

The final outcomes of this research provide valuable insights to the selection and design of neural network architectures based on their mathematical traits of the problem one is trying to solve. As the field of deep learning continues to improve, this thesis aims to guide all researchers and scholars in making informed decisions regarding the choice of neural network architectures for any given problem, thereby progressing the state-of-the-art in artificial intelligence and data science.

Table of Contents

List of Figures

List of Tables

.

Abbreviations

NN	**N**eural **N**etwork
ANN	**A**rtificial **N**eural **N**etwork
FC	**F**ully **C**onnected Neural Network
FFNN	**F**eed **F**orward **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
RNN	**R**ecurrent **N**eural **N**etwork
BiRNN	**Bi**-directional **R**ecurrent **N**eural **N**etwork
GRU	**G**ated **R**ecurrent **U**nits
GAN	**G**enerative **A**dversarial **N**etwork
DNN	**D**eep **N**eural **N**etwork
ReLU	**R**ectified **L**inear **U**nit
LSTM	**L**ong **T**erm **S**hort **M**emory
MSE	**M**ean **S**quared **E**rror
MAE	**M**ean **A**bsolute **E**rror
GPU	**G**raphical **P**rocessing **U**nit
CPU	**C**entral **P**rocessing **U**nit

Chapter 1

Introduction

1.1 Motivation & Overview

This thesis aims to provide a **"Comprehensive** and **systematic overview/analysis** of different types of **Neural Network architectures** with respect to their **efficiency** in solving different kinds of **Mathematical equations"**. These mathematical equations ranges from a simple equation (e.g. $[a + b = c]$) to a bit complex equations (e.g. $[\int a^2 cos(a) = b]$) in nature too. Hence, this thesis can act as a guide for proposing a novel framework or for selecting the most suitable Neural Network architecture for any problem given that we know the type of mathematical operations it might be performing.

As Neural networks are extremely powerful ML models that can be used to achieve many different objectives ranging from simple tasks like classification and regression to very complex tasks such as Natural Language Processing, image and speech recognition on the basis of input data provided.[Alzubaidi et al., 2021].

However, there are various different types of Neural Networks, for example:

1. fully-connected NNs (FC)
2. convolutional NNs (CNNs)
3. recurrent NNs (RNNs)
4. bi-directional RNNs (BiRNNs)
5. and many more.

each with their own set of rewards and drawbacks [Ye et al., 2019, Bianco et al., 2018].

Hence, selecting the best Neural Network architecture for any problem is not an easy task, as there are various factors affecting the following decision like the input data-set, available resources, desired precision and the compromise between complexity and performance [Cong and Zhou, 2023, Cordoni, 2020].

So, it becomes significant to evaluate and compare various architectures of Neural Networks on different problems and sets of input data, and to know their strengths and limitations, along with their similarities and differences.

1.2 Aims & Objectives

The aim of this project is to implement, compare and evaluate various types of Neural Network architectures on different Mathematical equations and equivalent synthetic data-sets, To propose a new framework for selection of the most fit architecture for any given problem.

The specific objectives for this project are as follows:

1. Conducting a detailed literature review on different existing neural network architectures and their real life applications.
2. Developing a standard evaluation framework/metric for comparing various architectures through multiple Mathematical Equations and synthetic data-sets.
3. Generating the training/testing data-sets for evaluation and training of the various neural network models.
4. Implementing and training different neural network architectures models on the set of generated data-sets.
5. Analyzing the results of each model and comparing these different architectures in terms of accuracy, performance, efficiency, and other metrics.
6. Summarizing all the different results and drawing inferences, contributing to the understanding of these different neural network architectures and the problems where they can be used.

Chapter 2

Related Work & Literature Review

2.1 Introduction

This chapter aims at exploring the in's and out's of the Neural Networks and their different architectures along with probing the generation of all the different synthetic data set required. It also focuses on critical evaluation of the earlier studies/research conducted in the field of Neural Networks and all the related work to help shape the research foundation.

2.2 Exploring Neural Networks

Neural networks are a specific type of machine learning models that try to replicate the learning mechanism of a biological organism in electronic medium. These models generally consists of many interconnected units known as neurons, they can process the data and transmit information through the networks with the help of internal variables known as weights, biases and activation functions.

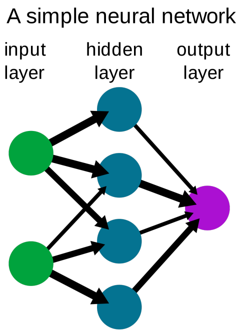


Figure 2.1: A simple Neural Network.

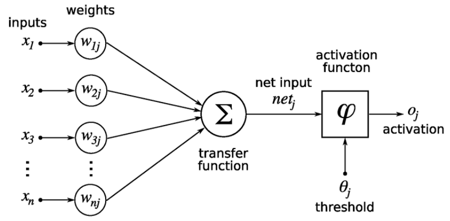


Figure 2.2: Illustration of neuron structure[Ren et al., 2018].

The key components of a basic NN model neurons' are as follows:

1. **Inputs**: These refer to the data that is fed into the NN model neuron from generally from outside source or either other neurons. Each input part generally has a weight if its coming from other neuron, which demonstrates how strongly the 2 neurons are connected.
2. **Weights**: These refer to how much relative contributions of each input to the neuron's output. Throughout the learning process, weights are changed to reduce the error between the expected output and the actual value.
3. **Bias**: This refers to the constant term that is always added to the sum of the weighted inputs. It enables the neuron to move the activation function[3.2.5] with respect to the input axis.
4. **Activation function**: The weighted sum of the inputs plus the bias are mapped to the neuron's output by this function it is generally non-linear in nature. Some examples of activation functions[3.2.5] are as follows: sigmoid, tanh, ReLU, and softmax.
5. **Output**: This refers to the value that a neuron produces following the application of the activation function. It might serve as a neuron's final output or as an input for any subsequent neuron.

These models can extract and learn even complex patterns from the provided data and are able to perform various types of tasks like object detection, natural language processing, image and speech recognition[Aggarwal, 2018].

Though NNs are imitations of the human brain / biological organism, still these are not exact copies of it. They are much simplified in nature and are abstract models that only capture basic properties of a neurological systems, such as learning from experience, adaption and self-organization. These models also have their own share of limitations and challenges like scalability, under-fitting, robustness, interpretability, over-fitting etc.[Ghorakavi, 2018].

NN models are generally trained by using many sets of expected input - expected output pairs. These sets of input-output values provides the external incentive for adapting/learning, and the NN models adjusts its internal variables such as weights and biases to rectify/minimize the total error between the predicted values and the actual values. Many different parameters and techniques are used to measure and improve the performance of neural networks like:

1. precision
2. accuracy
3. recall
4. confusion matrix
5. F1-score
6. regularization

and many more[Aggarwal, 2018, Caudill, 1989].
There are many different types of NN models depending upon the neuron configuration memory management and other factors. Some of these model types are as follows:

1. fully-connected

2. convolutional
3. recurrent
4. bi-directional
5. and many more.

each with their own rewards and limitations.

2.3 Historical perspective of Neural Networks

The history of Evolution of Neural Networks can be broken down in 5 important intervals.

1. **Conceptualisation [1940 - 1950]**:
   The idea of Neural Networks emerged around very early 1940's when the researchers tried to imitate the delicate workings of the Human Brain in electronic medium. The foundation of artificial neuron laid by neurophysiologist Warren McCulloch and mathematician Walter Pitts[McCulloch and Pitts, 1943].
2. **Innovation [1950 - 1960]**:
   Around late 1950's the idea of a simple single layered neural network model known as Perceptron with the capabilities of learning and binary classification based on the data provided as input was proposed by Frank Rosenblatt[Rosenblatt, 1958].
3. **Dark Era [1960 - 1970]**:
   During this time period the field of Artificial Neural Networks experienced a lot of setbacks due to limitations of Perceptron, especially in tackling complex issues. This resulted in doubt towards viability of Neural networks which in-turn resulted in lower funding and interest in the area[Minsky and Papert, 1969].
4. **Resurgence [1970 - 1990]**:
   Around starting of 1970's there came a very important breakthrough in field of ANN with an algorithm proposed by Paul Werbos known as Backpropogation but it wasn't until 1980's when researchers such as Geoffrey Hinton proposed the practical application of Backpropogation. Which kick-started the field of Artificial Neural Networks and Deep Learning[Werbos, 1988].
5. **Renaissance and the future[1990 - present]**:
   Driven by the leaps in computational power and the introduction of specialized hardware devices such as GPUs till 1990's. The field of deep learning and ANNs entered its renaissance period which pushed the field further, leading to amazing accomplishments in various fields like image recognition with convolutional neural networks (CNNs), Natural Language Processing, and other areas. Around 2010's we entered a trans-formative era for neural networks and deep learning. Due to availability of extremely large datasets with exceptionally powerful and fast GPUs and improvements in design and architecture of neural networks, like recurrent neural networks (RNNs), long short-term memory (LSTM) networks and Transformers, played a huge role in the success of ANNs on various fronts.

2.4 Different types/architectures of Neural Networks

As mentioned in section [1.1,2.2] there are various types of NN architectures and NN models based on different orientations of Neurons. In the following section we will discuss how different NN architectures respond to Mathematical problems. These are some of the different architectures which we will discuss with the relevant mathematical usage found in earlier conducted studies and research in the following section:

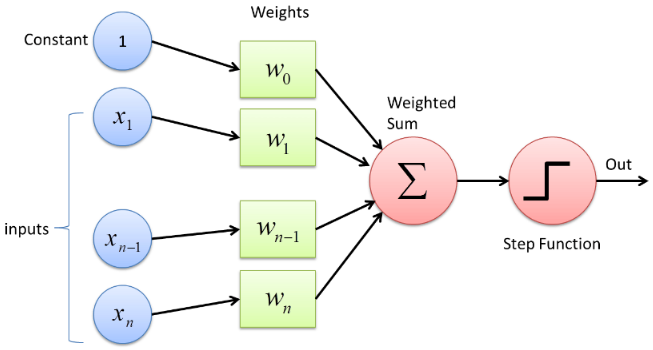
1. Perceptron
2. Feed-forward neural networks
3. Convolutional neural networks
4. Recurrent neural networks
5. Generative adversarial networks
6. Attention networks
7. Graph neural networks
8. Pre-trained state-of-the-art Architectures

2.4.1 Perceptron

A perceptron is a very basic neural network. It is often considered as the building block of ANNs. It was first proposed by Frank Rosenblatt around late 1950s

Working of Perceptron

A perceptron generally functions by taking multiple inputs which are binary in nature and adding them up after applying weights to them. Following this the sum is passed through an activation function[3.2.5] to produce the final output.



Source

Figure 2.3: Illustration of a Perceptron[Chowdhury, 2023].

Uses of Perceptron in Mathematical Problems

A perceptron on itself is not very powerful but it is still very efficient as a Machine Learning model. They are particularly fit for linearly separable problems, where a straight line/hyperplane can be drawn to differentiate the data of two classes.[Rosenblatt, 1958] It is useful in handling these kinds of Mathematical Problems:

1. **Classification**: These models are extremely efficient in performing Binary classification, where the aim is to differentiate between two classes of data.
2. **Regression**: These ML models are also very useful for regression tasks where the aim is to predict numerical data which is generally continuous in nature.

2.4.2 Feed-forward neural networks

A Feed-forward neural network also popularly known as multi-layer perceptron is also one of the classical neural network models in machine learning it is generally considered a much needed extension to the classic perceptron model[2.4.1] as it is more complex in nature because it contains an input layer, some hidden layers and an output layer. Where every hidden layer contains some amount of neurons which takes inputs of the sum of weighted outputs from the previous layer of neurons and pass the same weighted sum to the next layer.

Working of Feed-forward NN

This machine learning model generally works by trying to estimate the best mathematical function that could be fit on they provided input data set. Each individual node is a perceptron/neuron and they are composed of multiple neurons,They use

$$[Z_j = \sum_{i=1}^{n}(x_i, w_{ij}) + b_j(whereZ_j = weightedsum), A_j = \phi(Z_j)(A_j = activationfunction)] \tag{2.1}$$

formulae to calculate inputs to every input and hidden layer but unlike perceptron these ml models are not limited to only linearly separable problems but are also capable of understanding complex data too.

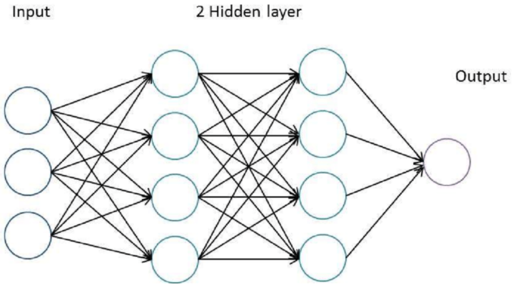


Figure 2.4: Illustration of a Feed-forward Neural Network.[Ma et al., 2019].

Uses of Feed-forward NN in Mathematical Problems

As feed forward neural networks are an upgrade to classical perceptron model. Hence, they are able to solve many more different types of Mathematical problems too. Some of them are as follows:

1. **Regression**: Continuing on the perceptron model these machine learning models are also able to perform regression on given input data[Goodfellow et al., 2016].
2. **Classification**: These ML models can perform both binary and multi-class classification depending upon type of input data provided[Bishop, 2007, LeCun et al., 2015].
3. **Optimization**: These models can also be used for finding the optimal points(maxima, minima) for the provided data.
4. **Time series**: These models find small success in time series prediction too[Zhang, 2003].
5. **Anomaly Detection**: These machine learning models find small success in the field of anomaly detection too[Chandola et al., 2009].

2.4.3 Convolutional neural networks

A convolutional neural network or a CNN are a very special type of deep learning algorithm that is generally applied to grid data such as pictures and videos. Hence, they have been extremely successful in many computer vision related tasks. But still, they are very useful in other fields such as machine translation and natural language processing too. They generally consist of convolution, pooling and flattening layers apart from the layers mentioned in Feed-forward NNs[2.4.2].

Working of CNN

CNN uses all the features mentioned in Feed-forward NNs[2.4.2] and has some additional features in form of convolution, pooling and flattening layers too.

1. **Convolution**: CNN uses this layer to filter/capture local patterns or features and complex structure in provided data using kernels/filters.
2. **Pooling**: This layer is used to down-sample the data and to focus on most prominent features captured with convolution layer.
3. **Flattening**: This is used to flatten data in a single dimensional vector.

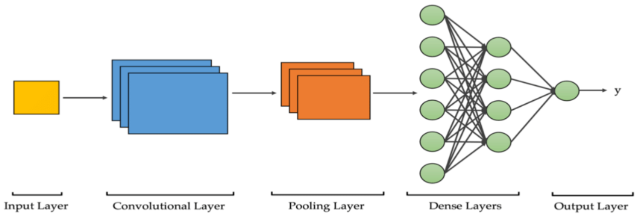


Figure 2.5: Illustration of a Convolutional Neural Network.[Sit et al., 2020].

Uses of CNN in Mathematical Problems

Apart from Vision and NLP problems CNN's have been used for many types of mathematical problems too. Some of them are as follows:

1. **Derivation**: These models can be used for derivation of formulas of mathematical principles[Caterini, 2017].
2. **Equation solving**: They can solve different types of problems such as quadratic linear and even trigonometric nature equations[Charton and Lample, 2022, Caterini, 2017].
3. **Mathematical symbol recognition**: A well-known use of these models is recognition of mathematical symbols/numbers that are handwritten[Navaneetha Krishnan et al., 2023].
4. **Proof verification**: One of the best uses of these models can be in verifying weather the provided mathematical proof is correct or not.

2.4.4 Recurrent neural networks

Recurrent neural network are special type of NN Machine Learning models as they are very well fit for handling data that is sequential in nature unlike normal Feed-Forward NNs[2.4.2]. Due to this ability of RNNs they gain ability to capture the long-range dependency in data over many different iterations of training.

Working of RNN

These neural network models have a feedback loop which allows them access the information from the previously provided data inputs because of this functionality it allows RNNs to identify the long range dependency in the input data. As RNNs have directed cycles due to way in which their neurons are connected. Which might cause the problem of gradient descent during Backpropogation. Hence to counter this issue these ml models use a modified version of back propagation algorithm known as back propagation through time this algorithm calculates the gradient for entire sequence of input data provided throughout time which helps in learning the temporal dependencies of the provided data[Cho et al., 2014].

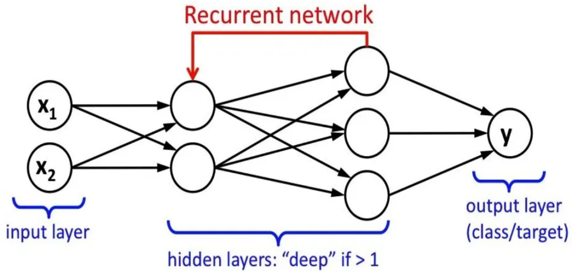


Figure 2.6: Illustration of a Recurrent Neural Network.[Valkov, 2018].

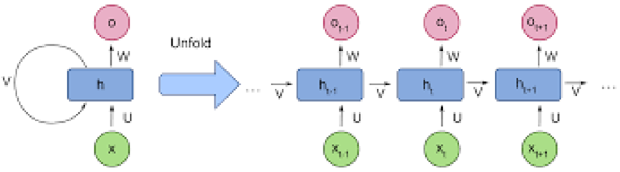


Figure 2.7: Unfurled RNN[fdeloche, 2017].

These ml models have a mechanism of hidden state which they update at every time step with respect to the current input data provided to the model. They use

$$h_t = f(x_t, h_{t-1})[Where(t = TimeStep), (h = HiddenState), (x = Data), (f = function)] \tag{2.2}$$

formulae to calculate hidden state for every input.

Different Types of Architecture in RNN

Depending on the different types of problems we can have many different types of RNN models. Some of these are as follows:

1. **Simple RNNs**: These models are generally useful in processing the data which is sequential in nature. These models are generally useful in processing the data which is sequential in nature, But they suffer from the very common vanishing gradient problem.
2. **LSTM**: These ml models were built to counter the problems faced by simple RNNs for the following reason they use modified version of back propagation and generally have memory cells which while making the architecture a little complex allows long short term memory models to access and store the current information and all previously processed data too. Due to this LSTM are generally slower in training and computation when compared with other RNN architectures[Hochreiter and Schmidhuber, 1997].
3. **GRU**: These ml models are also built to counter the issues which were faced by simple RNNs hence architecture wise they are very similar to LSTMs but they are much more simplistic in design. This quality of Gated Recurrent Units (GRUs) result in faster training and computation then LSTMs[Cho et al., 2014].

Uses of RNN in Mathematical Problems

RNN's have been used for many types of mathematical problems too. Some of them are as follows:

1. **Differential Equations**: These ml models are capable of solving different kinds of differential problems for varying orders which includes both linear and non-linear equations [Sanjeevi, 2018].
2. **Time Series**: As these models are extremely competent in handling sequential data. Hence they perform really well in prediction/time series analysis problems [Hochreiter and Schmidhuber, 1997].
3. **Generative Models**: These models find there use in generation of synthetic data too. This is not limited too mathematical data but also data such as music and text [Chung et al., 2015].

2.4.5 Generative adversarial networks

Generative adversarial networks (GANs) are a very novel and powerful approach in the field of Deep learning and neural networks which was proposed around 2014[Goodfellow et al., 2014] and very different from earlier discussed FFNNs[2.4.2], CNNs[2.4.3] and RNNs[2.4.4]. These neural networks generally contain 2 entirely different neural networks known as generators and discriminators. These ml models are generally good for generative tasks where the aim is to create new things while taking inspiration from the input data.

Working of Generative adversarial NN

These ml models generally work by putting 2 neural networks known as generators and discriminators in a competitive state against each other until the model converges. This learning procedure is known as adversarial training. Important parts/processes of a general GAN are as follows:

1. **Generator neural network**: The following part of the GAN aims to create data that resembles the actual data samples by taking random noise as the input. In the following case it will be mathematical equations or outputs[Goodfellow et al., 2014].
2. **Discriminator neural network**: The following part of the GAN aims to classify the outputs generated by generator networks in weather they are a real output of the provided data set or is generated by generator layer[Goodfellow et al., 2014].
3. **Adversarial training**: The following process refers to the iterative nature of feeding data for training purposes in a general adversarial network. In this both generator and discriminator neural networks are trained simultaneously with generator aiming at generating more realistic outputs when compared to the input data. While discriminator aiming in improving the classification between real and generated outputs with respect to the provided input data[Goodfellow et al., 2014].

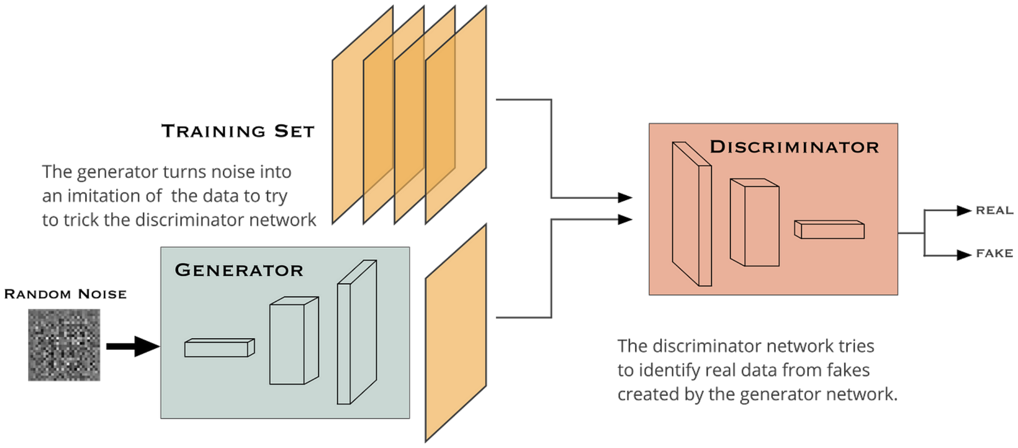

TRAINING SET
The generator turns noise into an imitation of the data to try to trick the discriminator network
RANDOM NOISE
GENERATOR
DISCRIMINATOR
REAL
FAKE
The discriminator network tries to identify real data from fakes created by the generator network.

Figure 2.8: Illustration of a Generative adversarial network.[Yalçın, 2023].

Uses of Generative adversarial NN in Mathematical Problems

GANs are still novel in nature. Hence, they are still part of many active researches and are investigated in many types of mathematical problems too. Some of them are as follows:

1. **Differential Equations**: These machine learning models shows promise in estimating solutions of differential equations ranging from simple to complex in nature due to Adversarial training.Hence, as the generator part of GAN gets better in coming up with Solutions that solve the equation at the same time the discriminator part of the GAN helps in weeding out solutions that are not realistic in nature[Randle et al., 2020].
2. **Optimization**: Following machine learning models show tremendous potential in optimization problems too. As these can be easily used for finding the global Optima for different types of mathematical equations ranging from simple single variable to multivariate complex problems too. The adversarial learning helps in finding global optimal solution while avoiding the problem of local optimal solution that plagues simple machine learning and neural network models[Cohen and Giryes, 2023].
3. **Data generation**: These machine learning models are excellent for data generation uses as the 2 parts of GANs are very useful in generating realistic data. As the discriminator neural network helps in making sure that data generated from generator neural network is off extremely high quality[Goodfellow et al., 2014].
4. **Super-Resolution**: Due to the generation part of these machine learning models they show tremendous potential in up-scaling images from low to high resolution [Ledig et al., 2017].

2.4.6 Attention networks

Attention based neural networks are an extension to preexisting ideas such as convolution neural networks[2.4.3], recurrent neural networks[2.4.4] and so on. These machine learning models introduce a new mechanism known as attention which allows the neural networks to focus their attention on a particular part of the supplied input data while producing the outputs. These models are mainly useful for natural language processing but are not limited to it.

Working of Attention based NN

As attention is a mechanism rather than a new type of neural network hence it is extensible to other existing approaches of neural network architectures Similar to RNNs they also make use of hidden states. It also makes use of a context vector which is the outcome of the mechanism itself. This context vector is helpful in calculating the weighted sum. Which is passed onto subsequent layers of neurons present after the attention mechanism. Some of the important processes/parts of the attention mechanism are as follows:

1. **Attention**: In this mechanism the machine learning model tries to focus on specific parts

of the input data provided while making the final prediction for the following process It assigns different weights to different parts of the input sequence based on importance of those specific parts in the final output.

2. **Context Vector**: This special vector captures the output of the attention mechanism which can be summarised as the weighted sum of the input provided where each weight focuses on specific parts of the input.
3. **Hidden State**: It functions similarly to its RNN[2.4.4] counterparts. Hence, they update at each step with respect to the current input data provided.

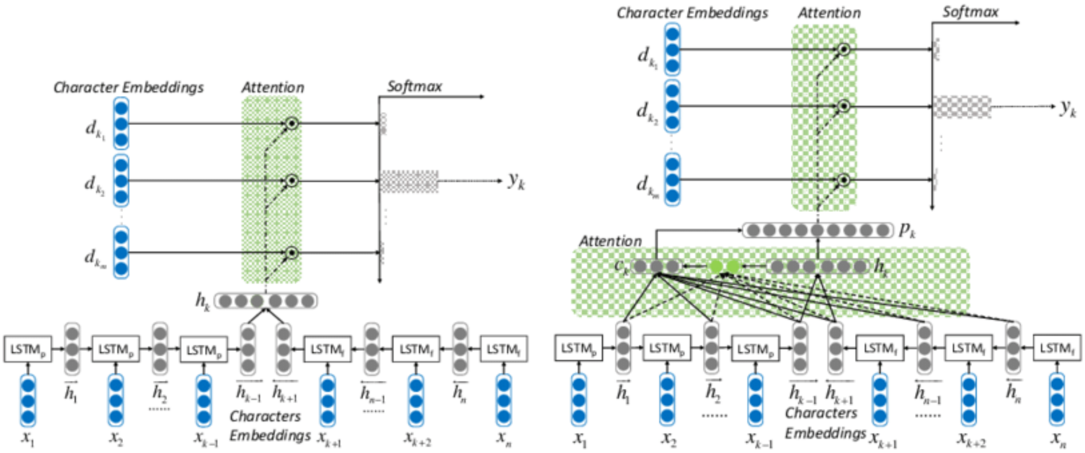


Figure 2.9: Illustration of an Attention based LSTM network.[Wang et al., 2019].

Uses of Attention based NN in Mathematical Problems

As attention is a mechanism rather than a modelling technique. Hence, it can be used with pre-existing architectures like CNN[2.4.3], RNN[2.4.4] to improve their mathematical applications. Some of the notable examples of this are as follows:

1. **Symbolic reasoning**: Attention based neural networks find moderate success in understanding and solving symbolic problems. Hence, they could be used for solving equations, proving theorems and many more activities examples of symbolic expressions are integrals and differential equations[Ericson and Jensfelt, 2022].
2. **Optimization**: Attention neural networks do extremely well in finding the global optimal solution many times even outperforming classical algorithms too. These machine learning models can achieve this due to focusing on the important parts of the input data provided[Cao et al., 2021].
3. **Data-driven modeling**: These machine learning models extremely useful for making predictions for new relationships based on existing mathematical relationships provided to them as an input. Hence making them useful for discovering previously unknown or overlooked relationships in the data[Vaswani et al., 2017].

2.4.7 Graph neural networks

These machine learning models add a new type of neural networks that are specially designed to work on graph data structures unlike simple FFNN[2.4.2]. These have gained some recognition in the fields of recommendation systems, network analysis and more. These types of machine learning models are still very new hence their full potential hasn't been explored till now and they are still part of a lot of ongoing research[Liu and Zhou, 2022].

Working of Graph neural network

A graph neural network generally works by converting simple input data into graph format where nodes denotes the independent and dependent variables present in an expression while the edges denote relationship that exists between these nodes. These edges represent the context between the dependent and independent variables[Zhou et al., 2020].
Graph neural networks can generally be considered as a special case of neural networks data design for specifically working on graph data. Due to this many of the aspects of a graph based neural networks are changed to better suit this scenario. Here are some of the aspects of a GNNs that are very particular to these types of machine learning models:

1. **Data Representation**: As discussed earlier too a graph neural network generally represents the data in the form of interconnected graphs where nodes of a graph represents all the dependent and independent variables while all the ages represents the existing relationship between these notes or variables.
2. **Node Embedding**: These machine learning models try to memorise the embedding for all the nodes which are present in the graphs. In an attempt to capture the existing relationship present between these nodes.
3. **Message Passing**: These machine learning models makes use of general message passing mechanism generally used in graph data structures for passing information from one node to another. The following process is done in an iterative manner to refresh their representation in the graph based on assimilated information.
4. **Special Convolution Function**: Just like normal CNNs[2.4.3] these graph based convolutional networks also have special functions/layers which are optimised for graph based data structures. Some of the prominent functions are as follows:
    (a) **Convolution Layer**: Like their simple convolutional network counterparts these convolutional layers are also here for the purpose of aggregating or assimilating the information from all the nearby nodes. A normal convolutional network is extremely adapted in handling grid-based data structures but these special convolution layers work well with the graph data structure.
    (b) **Pooling Layer**: Similar to the pooling operations that occurred in CNNs[2.4.3]. These graph based pooling layers are here for aggregating the graph structure present in these neural networks.
    (c) **Un-Pooling Layer**: Like the un-pooling operations that occurred in CNNs[2.4.3]. These graph based un-pooling layers are here for expanding the graph structure present in these neural networks.

5. **Special Attention Mechanism**: These machine learning models have their special attention mechanism too. As normal attention only works on simpler data structures hence a new attention mechanism that can work with graphs nodes was required. The following mechanism helps on focusing on the important nodes present in the graph while making the output predictions.

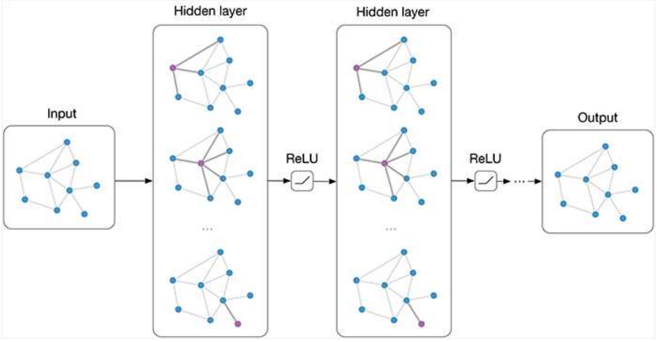


Figure 2.10: Illustration of a Graph based Neural Network.[Karagiannakos, 2020].

Uses of Graph neural network in Mathematical Problems

As GNNs are Graph based ml models hence they perform surprisingly well with mathematical problems too this can be attributed to their ability of representing input data as a graph and information aggregation techniques. Hence it could be used in many mathematical problems some of them are as follows:

1. **Optimization**: These machine learning models can make use of their graph structures and are able to find the global optima for many kinds of functions even problems with heavy and complex dependencies[Lei et al., 2022].
2. **Proof and Reasoning**: These machine learning models can easily analyse relationships that might exist between mathematical problems and their respective solutions. Hence, they are useful for theorem proving purposes. These can also be extended to derive new reasoning's based on existing knowledge[Paliwal et al., 2020].
3. **Equation Solving**: Due to the ability of representing input data as a graph these machine learning models are excellent for finding solutions for many different types of equations ranging from simple linear to differential equations [Liu and Xu, 2023, Liu and Zhou, 2022].

2.4.8 State-of-the-art Architectures

In the following section we will be discussing some state-of-the-art neural network architectures that are used for much more complex tasks such as image classification and text sentiment analysis rather than just solving mathematical problems. Some of these neural network architectures are as follows:

1. **CBAM (Convolutional Block Attention Module)**: CBAM is the straight up upgrade on the classic convolutional neural network architecture. It uses the concept of attention mechanism which increases its capability by not just focusing only on spatial data but also allowing it to apply channel wise attention[Woo et al., 2018].
2. **ConvNext**: ConvNext is a type of convolutional neural network which consists of multiple convolution layers, pooling layers, fully connected layers.
   This architecture shows considerable promise in image recognition and classification tasks[Younesi et al., 2024].
3. **ResNet50**: ResNet50 is a popular type of neural network architecture it is a variation on already existing ResNet architecture. The speciality of this model is that the following consists of 50 different layers. It also makes use of skip connection mechanism that helps the model how to circumvent the vanishing gradient problem[He et al., 2016].
4. **VGG16**: VGG16 is also an example of traditional CNNs.
   The specialty of the following is that it is very simplistic in nature and still manages to perform very effectively in image recognition tasks. It consists of total 16 layers which are comprised of convolution layers, max pooling layers and fully connected layers[Simonyan and Zisserman, 2014].
5. **AlexNet**: AlexNet is one of the award-winning architectures that is a variation of convolutional neural network architectures.
   It is comprised of total of five convolution layers, some Max pooling layers, and fully connected layers. It performs extremely well on image classification tasks, and it won an award in 2012 for the following too[Krizhevsky et al., 2012].

2.5 Synthetic Data Generation

Synthetic data is an important part of this project as we will be training all our different types of neural network architectures on the following generated data-sets. In this section we will discuss how will we be generating the data set for any mathematical problem.

For the purpose of generating synthetic data, we will be following a step-wise approach so we are able to understand, generate and validate our generated data. These steps in data generation are as follows:

1. **Defining/understanding the problem**:
2. **Choose a data generation method**:
3. **Validating the generated Data**:
4. **Adding complexity to the Data-set**:

2.5.1 Defining/understanding the problem

In the following step we try to identify the various parts of the mathematical problem that are needed for generation of synthetic data. The following parts might include these things:

1. **Identifying the domain of the given problem.**
   for example:-Trigonometric, Optimization and many more.
2. **Identifying the dependent and independent variables.**
   for example:- (y=mx+b) in which y is a dependent while x is an independent variable.
3. **Identifying the constants and assumptions in the problem.**
   for example:- (y=mx+b) in which b is a constants.
4. **Identifying the constraints in the given problem.**
   for example:- ReLU function has a constraint on numbers can't be less than zero.
5. **Identifying the range of data needed for the problem.**
   for example:- Range of a sigmoid function is generally from 0 to 1.
6. **Identifying the distribution of data needed for the problem.**
   for example:- Distribution of heights according to Gaussian distribution.
7. **Identifying the complexity of data needed for the problem.**
   for example:- Data required for a quadratic equation.

2.5.2 Choose a data generation method

In this step of synthetic data generation we try to choose a method of data generation based on different findings from the previous step. These methods can be further classified into 2 parts which are as follows:

1. **Based on Problem**: In the following scenario we are generally aware of the underlying mathematical problem and we can just substitute values to generate data. these can be further classified into 2 types:
   (a) **Equation Substitution**: In the following we try to generate new data points by just substituting new values in the underlying mathematical problem.
   (b) **Sampling**: In the following we try to generate a random sample of data based on the underlying mathematical distribution.
2. **Based on Data**: In the following scenario we are generally not aware of the underlying mathematical problem and we try to generate data on based of existing data-set.

2.5.3 Validating the generated Data

In this step we check the validity of the generated data set weather it follows the constraints, variables, range, complexity, and distributions that were found in the earlier steps of the process.

2.5.4 Adding complexity to the Data-set

After we have validated the data-set, we can try to add some complexity in the data so that it resembles a real life data-set. This can be done using the following mention techniques:

1. **Adding Noise**: Introducing some random noise to our generated data set Helps in creating a realistic data-set. As the data-set now contains varying level of imperfections and uncertainty too.
2. **Varying Difficulties**: We can add different level of complexity in our data-set, so it makes her model more robust and cover wider area in the search space for generalization of data.

2.6 Comparison and Evaluation of existing studies/research

In the following section we present a table to compare and evaluate different existing studies. To make it easier we will assess these studies on the following parameters:

1. **Applicability**
2. **Length**
3. **Usefulness**
4. **Implementation**
5. **Conclusion**

Papers related to Perceptron [2.4.1]

Paper Name	[Rosenblatt, 1958]
Applicability	High for its time but limited today
Length	Moderate
Usefulness	Conceptual significance
Implementation	Relatively simple
Conclusion	Landmark paper

Table 2.1: Evaluation of Papers related to Perceptron.

Papers related to Feed-forward neural network [2.4.2]

Paper Name	[Goodfellow et al., 2016]	[Bishop, 2007]	[LeCun et al., 2015]
Applicability	Wide	General	Specific
Length	Long (book)	Long (textbook)	Short (review paper)
Usefulness	Highly useful	Very useful	Very useful
Implementation	Moderately complex	Moderately complex	Low complexity
Conclusion	Essential reference	Valuable resource	Inspiring read

Table 2.2: Evaluation of Papers related to Feed-forward neural network part 1.

Paper Name	[Zhang, 2003]	[Chandola et al., 2009]
Applicability	Limited	Specific
Length	Short (journal paper)	Long (survey paper)
Usefulness	Moderately useful	Very useful
Implementation	Moderately complex	Moderately complex
Conclusion	Practical example	Important reference

Table 2.3: Evaluation of Papers related to Feed-forward neural network part 2.

Papers related to Convolutional neural network [2.4.3]

Paper Name	[Caterini, 2017]	**Charton & Lample,22**	**Navaneetha et al., 2023**
Applicability	Limited	Specific	Specific
Length	Long	Short	Moderate
Usefulness	Moderately useful	Moderately useful	Moderately useful
Implementation	Highly complex	Low complexity	Moderately complex
Conclusion	Theoretical foundation	Promising example	Practical comparison

Table 2.4: Evaluation of Papers related to Convolutional neural network.

Papers related to Recurrent neural network [2.4.4]

Paper Name	[Hochreiter and Schmidhuber, 1997]	[Cho et al., 2014]
Applicability	Wide	Specific
Length	Moderate	Moderate
Usefulness	Highly useful	Moderately useful
Implementation	Moderately complex	Moderately complex
Conclusion	Landmark paper	Practical example

Table 2.5: Evaluation of Papers related to Recurrent neural network part 1.

Paper Name	[Sanjeevi, 2018]	[Chung et al., 2015]
Applicability	Specific	Wide
Length	Short	Moderate
Usefulness	Moderately useful	Moderately useful
Implementation	Low complexity	Moderately complex
Conclusion	Introductory resource	Technical contribution

Table 2.6: Evaluation of Papers related to Recurrent neural network part 2.

....

Chapter 3

Requirements and Methodology

3.1 Introduction

In this section we try to summarise the proposed methodology for the thesis implementation. Along with functional, non-functional, software, and hardware requirements to implement the project.

3.2 Methodology

For the thesis to be considered successful the project should be implemented in a very planned manner. As there are many steps involved in the process of model selection and implementation for example:- Selection of problems, Selection of NN architectures, Implementation of the selected architectures and many more. If these are not done in a planned manner this might cause unexpected/unforeseen delays or hick ups.

To simplify this the project will have 6 major stages during the implementation phase these are as follows:

1. **Mathematical problem Selection** [3.2.1]
2. **Synthetic Data Generation** [3.2.2]
3. **Open-Source Dataset** [3.2.3]
4. **Neural Network Architecture Selection** [3.2.4]
5. **Neural Network Implementation** [3.2.5]
6. **Hyper-Parameter tuning** [3.2.6]
7. **Model Refinement** [3.2.7]

3.2.1 Mathematical problem Selection

Analysing and identifying a diverse range of mathematical equations/problems which range from extremely simple to a bit complex in nature these mathematical problems should cover various domains with ranging computational requirements.
The selection of these mathematical equations will be based on the following criteria:

1. **Complexity**: These mathematical problems/equations should vary from very simple to somewhat complex in nature. As this allows to test the full extent of capability of the implemented neural network model.
2. **Variation**: The included mathematical problems should come from diverse domains such as symbolic reasoning, calculus, optimization, and algebra. This is done to test the limits of the neural network architectures in the kinds of problems they can handle.
3. **Computational requirements**: The included mathematical problems should range in how much computationally heavy they are. This is done to test how efficient different architectures are with different types of mathematical problems.

3.2.2 Synthetic Data Generation

After selection of the mathematical problems that are required for the evaluation of different neural network architectures in the previous step[3.2.1]. We then try to generate large batches of data for training and testing purposes. The generated data will be specific to each mathematical problem selected in the previous step.
As synthetic data set is such an important part of the whole process. Hence generation of the data should also follow some basic guidelines to reduce the bias that might be involved with other preexisting data-sets. These guidelines are as follows:

1. **Encoding Mathematical Problems**: We should encode all the mathematical problems and their solutions in a format that is easily accessible and understood by the implemented neural network models.
2. **Diversity in generating data**: We should use multiple variations of parameters for generating these data sets. As this helps in creating different scenarios that could be possible with these different mathematical problems.
3. **Realism in generated data**: After generating the data we should check for the biases present in the data-set this helps in creating realistic test and training samples from the generated data.

3.2.3 Open-Source Datasets

As we will also be implementing some state-of-the-art neural network architectures for evaluation. Hence, we will also include 2 open-source and GDPR compliant datasets for analysing these models. As these architectures are generally preferred for extremely complex tasks, but we will still be analysing them on Image classification and test classification tasks. These datasets are as follows:

1. **IMDb movie review data set**: Text classification dataset.
2. **CIFAR 10**: Image classification dataset.

3.2.4 Neural Network Architecture Selection

Now we arrive at one of the most important steps for this thesis to be considered successful which is selection of various neural network architectures ranging from very simple to very specific and complex architectures which might require a lot of processing and computation power. These architectures will be selected on basis of how suitable they are for above selected mathematical problems[3.2.1] and how well they are able to understand the underlying relationship on the provided data [3.2.2]. Some of the examples are as follows:

1. **RNN** [2.4.4]
2. **CNN** [2.4.3]
3. and many more.

The architectures which show a promise in learning capabilities will be prioritised. Apart from this other architectures that are good at handling symbolic representation might be explored as well.

3.2.5 Neural Network Implementation

This is the step where we try to implement all these selected neural network architectures in the previous step for all the selected mathematical equations/problems and there synthetically generated data set. Followed by an initial round off evaluation of these implemented machine learning models.

The implementation of these neural network models also includes the designing phase for each of these ml models the following steps remain common:

1. **Layer Designing**: In the following step we decide how many inputs and how many number of neurons are present in each layer of the NN. Along with this we also design according to the specific needs of some models such as total number of attention layer (present in LSTMs[2.4.4]), pooling layer and convolution layer(present in CNNs[2.4.3]) present in the neural network.
2. **Activation function selection**: In this step we tried to decide what will be the activation function used by the model for fitting data that might be non-linear in nature. This helps in expanding the capabilities of the neural network and help in reducing the computational overload of the ml model.

   Some examples of Activation function are as follows:

   (a) **Sigmoid Function**: Generally used for the purpose of binary classification. Formula:

$$f(x) = 1/(1 + e^{-x})[Range(0, 1)] \tag{3.1}$$

(b) **ReLU (Rectified Linear Unit)**: An activation function which doesn't suffer from Vanishing Gradient issue and is very cost effective too. Formula:

$$f(x) = max(0, x)[Range(0, \infty)] \tag{3.2}$$

(c) **Tanh (Hyperbolic Tangent)**: Generally used in hidden layer neurons. It is very similar to Sigmoid Formula:

$$f(x) = (e^{x} - e^{-x})/(e^{x} + e^{-x})[Range(-1, 1)] \tag{3.3}$$

(d) and many more.

3. **Loss Function selection**: In the following step we will choose how we want to quantify the margin of error between the actual value add the output value produced by the ml model it is also known as loss function. Some examples of Loss function are as follows:

   (a) **Mean Squared Error (MSE)**: Its very useful in regression problems. Formula:

$$L(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.4}$$

   (b) **Binary Cross-Entropy**: Its very useful for binary classification. Formula:

$$L(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \tag{3.5}$$

   (c) **Categorical Cross-Entropy**: Its very useful in multi-class classification. Formula:

$$L(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{C} y_{ij}\log(\hat{y}_{ij}) \tag{3.6}$$

   (d) and many more.

3.2.6 Hyper-Parameter tuning

After implementing all the neural network models in the previous step now we try to optimise these models by tuning their hyper parameters or with modification in there architecture that might be required for the purpose of increasing their performance as much as we can.
Some of the Hyper-Parameters available for tuning are as follows:

1. **Learning Rate**: Learning great refers to the size of step at each iteration the optimization algorithm takes while adjusting the weights and biases of the neural network a lower learning rate leads to lower speed of convergence.
2. **Number of Hidden Layers/Neurons in Hidden Layers**: The following refers to the total number of hidden layers or the total number of neurons in a hidden layer present in a neural network increasing them might lead to learning more complex patterns, but it also increases the chance of over-fitting of data rather than learning pattern present in the data.

3. **Batch Size**: It refers to the total examples that are present in each batch of data used during many iterations of learning. Increasing it might lead to faster convergence, but a bigger batch size also requires much more memory.
4. and many more.

3.2.7 Final Model Selection

This is the final and one of the most important step of the implementation process. In the following step we finalise the machine learning model for each architecture with respect to each mathematical problem. Following this we train these selected ml models on the generated data-set how many times and we take the average of all these runs for the final evaluation process.

3.3 Requirements

In the following section we will discuss about the requirements that this thesis needs to fulfill to be considered successful.
As the scope of the project is quite big. Hence, it is necessary to define how can we measure that we were able to fulfil most of our desired goals for this project. So we have decided to split the requirements into 3 sections followed by a MoSCoW based summary of the requirements. These requirements are as follows:

1. **Functional** [3.3.1]
2. **Non-Functional** [3.3.2]
3. **Implementation** [3.3.3]

3.3.1 Functional Requirements

The following section refers to the most important requirements that the project needs to fulfil to be considered successful. These requirements form the very basis for a complete and overall analysis for "Comparison and Evaluation of different neural network architectures" with respect to mathematical problems. These functional requirements are as follows:

1. **Selecting the Mathematical problems**
2. **Generating the required data for the selected problems**
3. **Selecting the different Neural Network Architectures**
4. **Implementing the selected Neural Network Architectures**
5. **Training and Evaluating the implemented models**
6. **Analysing, comparing and summarizing all the results**

3.3.2 Non-Functional Requirements

This section refers to those requirements that are good to have and can enhance the quality of the final thesis. But these requirements are not necessary to be fulfilled for an overarching analysis or even comparing different types of neural network architectures and generation of the final report. These requirements are as follows:

1. **Interpretability of the Neural network**
2. **Computational efficiency**
3. **Ease of use**
4. **Result reproducibility**
5. **Novel approaches by combining different preexisting approaches**

3.3.3 Implementation Requirements

This section refers to the requirements that are needed for implementation of the project thesis itself. These can be divided into 2 separate parts. These parts are as follows:

1. **Software Requirements**
2. **Hardware Requirements**

Software Requirements

As far as software is required, we will need to choose a programming language as the base language. For the following project Python seems to be a good starting point as it has good support available online for debugging purposes meanwhile also many data science packages which play an important role in implementation of these machine learning models are also readily available. Apart from these, we might also need some tools for visualising the outputs of our models too. In the following scenario too, Python provides an excellent support with many visualization libraries easily available in market. For tracking the progress of the project, we will use GitHub as our version control system.

Summary and examples of the software tools needed for project implementation:

1. Programming Language such as: Python
2. Deep learning libraries such as: TensorFlow, PyTorch
3. Mathematical libraries such as: NumPy, SciPy
4. Visualization libraries such as: Matplotlib, Seaborn
5. Version control system such as: GitHub, BitBucket

Hardware Requirements

For implementation of the following thesis project, we will need a high-performance system preferably with the GPU for efficient training of these different machine learning models. Along with high storage capacity for storing generated data sets and the output of these machine learning models.

Summary of the hardware tools needed for project implementation:

1. High-performance CPU
2. Sufficient RAM
3. Dedicated GPU
4. Adequate storage space

3.3.4 MoSCoW based summary of the requirements

N°	**Feature Description**	**MoSCoW**	**Priority**
1	A good selection of Mathematical problems.	M	H
2	Generation of the required data for the selected problems.	M	H
3	A selection of many different Neural Network Architectures	M	H
4	Implementation, Training and Evaluation of the selected NN	M	H
5	Analysis, comparison and summary of all the results.	M	L
6	Tuning of different Neural Network Architectures	S	L
7	Ease of use	S	M
8	Interpretability of the Neural network	C	L
9	Result reproducibility	C	L
10	Novel approaches	C	M
11	Comparison with other ML Models	W	L
12	Testing scalability of the Neural Network Architectures	W	L

Table 3.1: MoSCoW based summary of the requirements

Chapter 4

Professional, Legal, Ethical and Social Issues

4.1 Introduction

In the following section of the report, we will discuss about the various sociological aspects of the thesis including professional issues legal issues ethical issues project planning and much more. This should provide a comprehensive outlook on the implementation, literature review and various other social aspects of the project itself.

4.2 Professional Issues

The work done on the project will adhere to all the professional standards and practices in the field and the code written will adhere to the British Computing Society (BCS) code of conduct. All the code will be written to a high standard of understanding with comments present throughout for the necessary and needed clarity. There will be adequate documentation present for all the different project related activities conducted during the thesis research. All libraries,third-party software, and other services shall only be used if their licenses let it. Any citations or outside data would be properly referenced for the needed clarity.

4.3 Legal Issues

The following research will adhere to all the laws/regulations that are relevant. During the work on the following, many existing external libraries and program products might be used for the success of the research. The use of all these external libraries and software will be under strict adherence to the licensing restrictions of their respective authorities.

As the following research uses only standard and GDPR compliant external data-set apart from the synthetic/artificially generated data-set and any personal, confidential and sensitive data isn't required for the aims of training or evaluation of the project. Hence, their shouldn't be any legal issues from the implementation point of view.

4.4 Ethical and Social Issues

As the data used in the project is mostly synthetic in Nature and could be produced using a simple scripts in most of the cases and as any personal, confidential and sensitive data is not required for the aims of training or evaluation of the project in general. Hence, the project doesn't raise any ethical and social concerns in itself but it does contribute to the question of "How improving the artificial neural networks capability of learning can affect how any general school teaches a student and does their evaluation of the subject knowledge?".

Chapter 5

Implementation

5.1 Introduction

In the following section we will discuss the implementation of various synthetic datasets, neural network architectures and the evaluation matrix[6.1] for the following machine learning models

5.2 Generation of the synthetic data

5.2.1 Overview

Here we will try to discuss the implementation and the crucial role synthetic data plays in training of these machine learning models. Building upon the methodology[3.2.2] of how we will generate the synthetic data. For training the neural network architectures that are implemented for the project we generated synthetic data for a variety of mathematical problems ranging from easy to little bit complex in nature. These problems can be broadly classified into 2 categories these are as follows:

1. **Simple Arithmetic/mathematical operations**: These operations are representing various small mathematical problems that are really simplistic in nature.
2. **Statistical distribution functions**: These operations are a variety of data spread functions which are statistical in nature.

Some examples of these problems are as follows:

1. **Addition**: It contains three columns: var1, var2 and result.

$$Formula : f(result) = var_1 + var_2 \tag{5.1}$$

2. **Subtraction**: It contains three columns: var1, var2 and result.

$$Formula : f(result) = var_1 - var_2 \tag{5.2}$$

3. **Multiplication**: It contains three columns: var1, var2 and result.

$$Formula : f(result) = var_1 \times var_2 \tag{5.3}$$

4. **Exponential**: It contains three columns: var1, var2 and result.

$$Formula : f(result) = var_1^{var_2} \tag{5.4}$$

5. **Trigonometrical**: It contains 2 columns: var1 and result.

$$Formula : f(result) = trig\_function(var_1)[trig\_function : sin, cos] \tag{5.5}$$

6. **Normal distribution**: It contains 2 columns: var1 and result.

$$Formula : f(result) = var_1 \sim \mathcal{N}(\mu, \sigma^2) \tag{5.6}$$

7. **Exponential distribution**: It contains 2 columns: var1 and result.

$$Formula : f(result) = var_1 \sim \exp(\lambda) \tag{5.7}$$

and various types of other datasets too.
As generating these synthetic data sets involve Various mathematical problems, functions distributions so to create a very realistic artificial data set We will be using Python programming language to implement the script that will be generating these data sets.

5.2.2 Requirements

As already discussed in Implementation requirements section[3.3.3]. Here is a Summary of the software tools and hardware requirements that are used for generation of Synthetic Data:

1. Programming Language: Python
2. Mathematical libraries: NumPy, SciPy
3. Version control system: GitHub
4. High-performance CPU
5. Sufficient RAM
6. Dedicated GPU
7. Adequate storage space

5.2.3 Implementation and examples

As implementation of each synthetic data set follow a mathematical problem each. Hence, we will try to make a dedicated Python function that generates the synthetic data common type of synthetic data will be generated using the same Python function with minor tweaks. While vastly different types of data set will be generated using dedicated Python functions. These functions will generally try to follow a common structure throughout with inclusion of problem specific variations.
Here is a general outline of the Python function.

```
#importing needed libraries
import numpy
import pandas
import io
import math
import scipy.stats

#setting up constants
total_dataset_size = integer
'''
Problem specific constants
'''
#file operation
file_name="Problem_specific_dataset"
file_location='./../Data/'
file_type='.csv'
is_file_index_needed=False

#data generation
number_of_iteration = total_dataset_size
result_list=[]
for index in range(number_of_iteration):
    intermediate_list=[]
    '''
    Problem specific processing
    '''
    result_list.append(intermediate_list)

#data frame creation and saving created datasets
df=pandas.DataFrame(result_list)
df=df.rename(columns={0: '''
    Problem specific column creation
    '''})
df.to_csv(file_location+file_name+file_type,
index=is_file_index_needed)
```

Some examples of these python functions are as follows:

1. Addition data set:

   ```
   #importing needed libraries
   import numpy
   import pandas
   ```

```
import io
import math
import scipy.stats

#setting up constants
total_dataset_size = 20000
#file operation
file_name=”addition_dataset”
file_location=’./../Data/’
file_type=’.csv’
is_file_index_needed=False

#data generation
number_of_iteration = total_dataset_size
result_list=[]
for index in range(number_of_iteration):
    intermediate_list=[]
    interim=index
    train_1=interim
    train_2=interim+1
    test=train_1+train_2
    intermediate_list.append(train_1)
    #intermediate_list.append(train_2)
    intermediate_list.append(test)
    result_list.append(intermediate_list)

#data frame creation and saving created datasets
df=pandas.DataFrame(result_list)
df=df.rename(columns={0: ”var_1”,1: ”var_2”, 2: ”result”})
df.to_csv
(file_location+file_name+file_type
,index=is_file_index_needed)
```

2. Normal distribution data set

```
#importing needed libraries
import numpy as np
import pandas as pd
from scipy.stats import norm

#Setting up constants
total_dataset_size = 20000
normal_mean = 0
normal_std = 1
#file operation
```

```
file_name=”normal_distribution_dataset”
file_location = ’./../Data/’
file_type = ’.csv’
is_file_index_needed=False

def generate_normal_dataset
(num_samples, normal_mean, normal_std):
    normal_data = norm(loc=normal_mean,
    scale=normal_std)
    .rvs(size=num_samples).reshape(-1, 1)
    # Generating random binary labels
    labels = np.random.randint
    (2, size=num_samples).reshape(-1, 1)
    df = pd.DataFrame
    (np.concatenate([normal_data, labels], axis=1)
    , columns=[’var_1’, ’result’])
    return df
df = generate_normal_dataset
(total_dataset_size, normal_mean, normal_std)
df.to_csv
(file_location+file_name+file_type
,index=is_file_index_needed)
```

5.2.4 Benefits

The data being synthetic in nature for these machine learning models provides us with various advantages while training these machine learning models. Some of these are as follows:

1. Generation of perfect datasets without noise.
2. Greater control over several factors and variables that are needed in these mathematical problems.
3. Control over total number of rows needed in these data set.
4. Control over the range in which data is generated.

and many more.

5.2.5 Challenges

As synthetic data provides numerous benefits in terms of training the machine learning models that are implemented during this thesis project it also comes with its fair share of problems. These problems should be considered and addressed when generating these synthetic data sets. Some of these problems are as follows:

1. Insurance of data quality.
2. Maintenance of diversity in the synthetic data.
3. High computational requirements for generating the synthetic data.

and many more.

5.3 Implementation of neural network architectures

5.3.1 Overview

In the following section we will try to discuss the implementation of various neural network architectures that are evaluated for the thesis building upon the methodology[3.2.5] of how we will design, train, test and tweak these machine learning models. As the core focus of the thesis is on comparison and evaluation of distinct types of neural network architectures. Hence, we have chosen a wide and diverse variety of architectures ranging from extremely basic and fundamental approaches to some of the recent advance state of the art architectures.

Though already discussed in literature review section[2.4] here is the list of several architectures and their summary of these architectures that are implemented for the success of the project:

1. **Perceptron[2.4.1]**: The following architecture consist of a single layer of neuron and a simple activation function these kinds of machine learning models are useful for simple classification/regression tasks but are extremely limited in their capabilities.
2. **Feed forward NN[2.4.2]**: As already discussed in the literature review the following architectures is an upgrade on the classic perceptron with multiple layers containing multiple neurons, they are better at understanding the underlying relation present in the data.
3. **Convolutional Neural Networks[2.4.3]**: This architecture specially excels in finding the relation between spatial features Data present in very structured form of data they use the concept of down sampling and pulling the data with the help of specific layers of neurons such as convolution layer and pooling layer. Though these models are extremely successful capturing spatial relationship they struggle with data that lacks innate special structures.
4. **RNN[2.4.4]**: These kinds of neural network models find great success in processing data which is sequential in nature making them a great fit for sequence prediction and time forecasting tasks. In a similar fashion to convolutional neural networks these models also struggle in the scenario where data is not sequential in nature.
5. **LSTM[2.4.4]**: As these neural network architectures are an upgrade on regular recurrent neural network architecture as they address the problem of vanish ingredient.Hence, these neural network models also find great success in processing data which is sequential in nature. They also struggle like recurrent neural networks as these models are also not great in the scenario where data is not sequential in nature.
6. **GAN[2.4.5]**: The following kind of neural network model consists of two different neural network models known as a generator and a discriminator. These models can be used for

generating synthetic data if needed as they try to perfect type of data that is generated while also improving simultaneously.

7. **Transformer**[2.4.6]: They are a new kind of neural network architecture that has shown tremendous promise in the fields of natural language processing problems. They generally utilise the process of self-attention how to identify the relationship between data and the output. Hence, they are extremely well with sequential data and can perform similar task as recurrent neural networks. As they perform very similarly to recurrent neural networks they also struggle with the scenario where data is not sequential in nature.
8. **Advanced architectures**: We will also be implementing some state-of-the-art neural network model architectures[2.4.8]. Though these architectures are generally preferred for extremely complex tasks, but we will still be analysing them on Image classification and test classification tasks. For the following we will be using 2 open-source GDPR compliant datasets[3.2.3].
   Below is the list of these implemented state-of-the-art architectures:
   (a) **CBAM (Convolutional Block Attention Module)**: The following kind of neural network architecture is a recent progress in the case of traditional convolutional networks as it uses the mechanism of attention how to upgrade the convolutional capabilities of the CNNs this helps in identifying the features that are consistently present and discarding the features that are unconnected in nature. Hence, these models are successful in object detection and image classification problems[[Woo et al., 2018]].
   (b) **ConvNext**: The following kind of neural network architecture is an upgrade on regular convolutional network architecture. It uses a variety of new techniques for the purpose of achieving higher efficiency and accuracy in extracting spatial features from structured data for example: - object detection and image recognition. It achieves this by trying out several types of convolution layers and introducing new techniques such as skip connections in the regular convolutional neural network architecture[[Younesi et al., 2024]].
   (c) **ResNet50**:The following kind of architecture is a popular variation on the pre-existing residual network architecture. As the residual network architecture is already famous for its extremely high performance in image classification tasks and transfer learning. Hence, this variation enables did you do network to form deep connections as it consists of fifty layers. Because these neural network models use residual connections and skip connections it makes it easy for such machine learning models to circumvent the vanishing gradient problem[[He et al., 2016]].
   (d) **VGG16**: The following neural network model architecture is an amazingly simple straightforward and a classic example of convolutional network architecture. It consists of sixteen convolutional fully connected layers. Despite its considerably basic architecture it has been a staple of successful neural network models as it boasts high performance in image classification problems[[Simonyan and Zisserman, 2014]].
   (e) **AlexNet**: The following neural network architecture Is considered a breakthrough in both convolutional neural networks and the field of deep learning itself. It generally consists of 8 layers Which are of 3 types convolution layers, max pooling layers and fully connected layers[[Krizhevsky et al., 2012]].

5.3.2 Requirements

As already discussed in Implementation requirements section[3.3.3]. Here is a Summary of the software tools and hardware requirements that are used for training and testing these Neural Network Architecture:

1. Programming Language: Python
2. Deep learning libraries: TensorFlow
3. Mathematical libraries: NumPy, pandas
4. Version control system: GitHub
5. High-performance CPU
6. Sufficient RAM
7. Dedicated GPU
8. Adequate storage space

5.3.3 Implementation and examples

As implementation of each neural network architecture can be vastly different from each other. Hence, we will try to use common programming language and libraries to generate these machine learning models. So, there is a uniformity among different machine learning models, and they can be just fairly. Apart from this in the following section we will provide an example of the layer configurations for all the implemented neural networks in this thesis.

Here is the list of several architectures and their example of layer configuration that are implemented for the success of the thesis:

1. **Perceptron**: optimizer= adam, loss= mean squared error, metrics= accuracy

```
#Layer configuration
    Perceptron = tf.keras.Sequential([layers.Dense(1)])
```

2. **Feed forward NN**: optimizer= adam, loss= mean squared error, metrics= accuracy

```
#Layer configuration
    MultiLayerPerceptron = tf.keras.Sequential(
    [layers.Dense(64, activation='linear ',
    input_shape=(dset_features.shape[1],)),
    layers.Dense(32, activation='selu '),
    layers.Dense(1, activation='linear ')])
```

3. **Convolutional NN**: optimizer= adam, loss= mean squared error, metrics= mae

```
#Layer configuration
    Convolutional = tf.keras.Sequential([
    layers.Conv1D(32, kernel_size=2,
    activation='relu ',
          input_shape=(dset_features.shape[1], 1)),
          layers.Flatten(),
          layers.Dense(64, activation='relu'),
          layers.Dense(1, activation='linear')])
```

---

4. **RNN**: optimizer= adam, loss= mean squared error, metrics= mae

```
#Layer configuration
          RNN = tf.keras.Sequential([
          layers.SimpleRNN(32, activation='relu',
          input_shape=(dset_features.shape[1], 1)),
          layers.Dense(64, activation='relu'),
          layers.Dense(1, activation='linear')])
```

---

5. **LSTM**: optimizer= adam, loss= mean squared error, metrics= mae

```
#Layer configuration
          LSTM = tf.keras.Sequential([
          layers.LSTM(32, activation='relu',
          input_shape=(dset_features.shape[1], 1)),
          layers.Dense(64, activation='relu'),
          layers.Dense(1, activation='linear')])
```

---

6. **GAN**: optimizer=Adam(learning rate=0.0002, beta 1=0.5), loss= binary cross-entropy, metrics= accuracy

```
#Layer configuration
          # Define generator model
          def build_generator(latent_dim):
               model = Sequential()
               model.add(Dense(4 * 4 * 256, input_dim=latent_dim))
               model.add(LeakyReLU(alpha=0.2))
               model.add(Reshape((4, 4, 256)))
               model.add(Conv2DTranspose(128,
               kernel_size=4, strides=2, padding='same'))
               model.add(LeakyReLU(alpha=0.2))
               model.add(Conv2DTranspose(128,
               kernel_size=4, strides=2, padding='same'))
               model.add(LeakyReLU(alpha=0.2))
               model.add(Conv2DTranspose(3,
               kernel_size=4, strides=2,
               padding='same', activation='tanh'))
               return model

          # Define discriminator model
          def build_discriminator():
```

```
    model = Sequential()
    model.add(Conv2D(64, kernel_size=3, strides=2,
    padding='same', input_shape=(32, 32, 3)))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Conv2D(128, kernel_size=3, strides=2,
    padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Conv2D(256, kernel_size=3, strides=2,
    padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))
    return model

# Define GAN model
def build_gan(generator, discriminator):
    discriminator.trainable = False
    model = Sequential()
    model.add(generator)
    model.add(discriminator)
    return model
```

7. **Transformer**: optimizer= adam, loss= Sparse Categorical Cross-entropy, metrics= accuracy

```
#Layer configuration
    class TransformerModel(tf.keras.Model):
        def __init__(self, num_classes,
        embed_dim=32, num_heads=2, ff_dim=32, dropout=0.1):
            super(TransformerModel, self).__init__()
            self.embedding_layer = tf.keras.
            layers.Embedding(input_dim=max_features,
            output_dim=embed_dim)
            self.transformer_block = TransformerBlock
            (embed_dim, num_heads, ff_dim, rate=dropout)
            self.global_average_pooling =
            GlobalAveragePooling1D()
            self.dropout = Dropout(dropout)
            self.dense = Dense
            (num_classes, activation='softmax')
```

8. **CBAM (Convolutional Block Attention Module)**: optimizer= adam, loss= Sparse Categorical Cross-entropy, metrics= accuracy

```
#Layer configuration
    class CNNWithChannelAttention(tf.keras.Model):
        def __init__(self):
            super(CNNWithChannelAttention, self)
            .__init__()
            self.conv1 = tf.keras.layers.
            Conv2D(32, (3, 3), activation='relu',
            input_shape=(32, 32, 3))
            self.maxpool1 = tf.keras.layers.
            MaxPooling2D((2, 2))
            self.conv2 = tf.keras.layers.
            Conv2D(64, (3, 3), activation='relu')
            self.maxpool2 = tf.keras.layers.
            MaxPooling2D((2, 2))
            self.flatten = tf.keras.layers.Flatten()
            self.fc1 = tf.keras.layers.
            Dense(64, activation='relu')
            self.fc2 = tf.keras.layers.
            Dense(10, activation='softmax')
            self.channel_attention = ChannelAttention()
```

9. **ConvNext**: optimizer= adam, loss= Sparse Categorical Cross-entropy, metrics= accuracy

```
#Layer configuration
    convnext_model = tf.keras.Sequential([
        tf.keras.layers.Conv2D(32, (3, 3),
        activation='relu', input_shape=(32, 32, 3)),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.Conv2D(64, (3, 3),
        activation='relu'),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.Conv2D(64, (3, 3),
        activation='relu'),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(64,
        activation='relu'),
        tf.keras.layers.Dense(10,
        activation='softmax')])
```

10. **ResNet50**: optimizer= adam, loss= Sparse Categorical Cross-entropy, metrics= accuracy

```
#Layer configuration
    resnet_model = tf.keras.applications.
    ResNet50(weights=None,
    input_shape=(32, 32, 3), classes=10)
```

11. **VGG16**: optimizer= adam, loss= Sparse Categorical Cross-entropy, metrics= accuracy

```
#Layer configuration
    vgg_model = tf.keras.applications.
    VGG16(weights=None, input_shape=(32, 32, 3),
    classes=10)
```

---

12. **AlexNet**: optimizer= adam, loss= Sparse Categorical Cross-entropy, metrics= accuracy

```
#Layer configuration
    def create_alexnet(input_shape, num_classes):
        model = tf.keras.Sequential([
            tf.keras.layers.Conv2D(64,
            (3, 3), strides=(1, 1), padding='same',
            activation='relu', input_shape=input_shape),
            tf.keras.layers.MaxPooling2D
            (pool_size=(2, 2), strides=(2, 2)),
            tf.keras.layers.Conv2D
            (192, (3, 3), padding='same',
            activation='relu'),
            tf.keras.layers.MaxPooling2D
            (pool_size=(2, 2), strides=(2, 2)),
            tf.keras.layers.Conv2D
            (384, (3, 3), padding='same',
            activation='relu'),
            tf.keras.layers.Conv2D
            (256, (3, 3), padding='same',
            activation='relu'),
            tf.keras.layers.Conv2D
            (256, (3, 3), padding='same',
            activation='relu'),
            tf.keras.layers.MaxPooling2D
            (pool_size=(2, 2), strides=(2, 2)),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(4096,
            activation='relu'),
            tf.keras.layers.Dropout(0.5),
            tf.keras.layers.Dense(4096,
            activation='relu'),
            tf.keras.layers.Dropout(0.5),
            tf.keras.layers.Dense(num_classes,
            activation='softmax')])
        return model
```

5.3.4 Challenges

As the project aims to implement a lot of neural network models. Each model being widely different from how other model is implemented. Hence, this ambitious task also comes with its particularly unique problems too. Which were considered and addressed during implementation of these machine learning models. These problems are as follows:

1. Complexity of the neural network model
2. Hyper parameter optimization
3. High computational requirements

5.4 Implementation of the evaluation matrix

5.4.1 Overview

In the following section we will discuss about the implementation of evaluation matrix discussed in detail in [6] which utilises various parameters to evaluate the implemented machine learning models during this thesis. As the evaluation of these neural networks require these models to be tested on many different parameters hence, we came up with A comprehensible matrix that measures all these parameters for each of these machine learning models.

These evaluation matrix will consist all the following parameters mentioned below:

1. Accuracy
2. MSE
3. MAE
4. RMSE
5. F1 Score
6. Training Time
7. Inference Time
8. Interpretability
9. Generalization
10. Robustness
11. Suitability for NN Architecture

these parameters are discussed in detail in [6].

5.4.2 Requirements

As already discussed in Implementation requirements section[3.3.3]. Here is a Summary of the software tools and hardware requirements that are used for Evaluation:

1. Programming Language: Python

2. Mathematical libraries: NumPy, pandas
3. Version control system: GitHub
4. High-performance CPU
5. Sufficient RAM
6. Adequate storage space

5.4.3 Implementation and examples

Like implementation of each synthetic data set these following evaluation matrix for machine learning models also follow very straight forward mathematical problem. Hence, we will try to make a dedicated Python function that generates all these matrix parameters These models are being trained for being evaluated.
Here is a general outline of the Python function that will generate these matrix parameters.

```
def train_model(model, X_train, y_train, X_test, y_test,
model_name, dataset_name, total_epoch=50):
    tensorboard_callback =
    TensorBoard(log_dir='./'+model_name+dataset_name+'logs ')
    start_time = time.time()
    history = model.fit(X_train, y_train, epochs=total_epoch,
    validation_data=(X_test, y_test), callbacks=
    [tensorboard_callback])
    training_time = time.time() - start_time
    predictions = model.predict(X_test)
    return model, history, predictions, training_time

def test_model(model, X_test, y_test, model_name,
dataset_name):
    start_time = time.time()
    predictions = model.predict(X_test)
    inference_time = time.time() - start_time
    metrics = calculate_metrics(predictions, y_test)
    metrics['Training Time'] = training_time
    metrics['Inference Time'] = inference_time
    metrics['Model type '] = model_name
    metrics['dataset '] = dataset_name
    metrics_list = list(metrics.values())
    pd.DataFrame([metrics_list],
    columns=metrics.keys()).
    to_csv('./'+model_name+dataset_name+'metrics.csv ',
    index=False)
    return metrics

def calculate_metrics(predictions, true_labels,threshold):
```

```
correct_predictions = np.sum(np.abs(true_labels -
predictions.flatten()) <= threshold)
total_predictions = len(true_labels)
accuracy = correct_predictions / total_predictions
mse = mean_squared_error(true_labels, predictions)
mae = mean_absolute_error(true_labels, predictions)
rmse = np.sqrt(mse)
f1 = f1_score(true_labels.round(),
predictions.round(), average='micro')
return {'Accuracy': accuracy, 'MSE': mse, 'MAE': mae,
    'RMSE': rmse, 'F1 Score': f1}
```

Chapter 6

Evaluation

6.1 Introduction

In the following section we will try to discuss the importance of evaluation of the implemented neural network architectures and answer the question of "how should we evaluate these models in a systematic and detailed way?"

Evaluating these machine learning models provides us with 3 very important benefits these are as follows:

1. **Optimizing performance**: Evaluating the neural networks models helps us in optimising the performance of the said machine learning models tuning the hyper parameters so we can get the best result for any neural network model for the said mathematical problem.
2. **Troubleshooting**: Evaluating these machine learning models also helps us to troubleshoot these neural networks by analysing the common problems that might occur during the training of these models such as:
   (a) **Overfitting**: It is the situation in which a model memorises the data rather than learning the underline process from the data during training.
   (b) **Underfitting**: It is the situation in which a model didn't get enough data or time to identify the underlying process from the data during training.
   and many more common problems too.
3. **Comparison**: As the title of the thesis ("Comparison and Evaluation of Neural Network Architectures") suggests Comparison is a big part of the following research and without a good evaluation mechanism it will not be possible. As evaluation provides us with quantifiable figures that help in identifying whether or not the machine learning model is actually performing well on the given problem.

6.2 Evaluation Parameters

For the overall evaluation of the models implemented during the thesis project itself we will try to follow the standard machine learning model evaluation parameters. These parameters can be classified in to 2 types known as Quantitative[6.2.1] and Qualitative[6.2.2] parameters. These evaluation parameters contains various metrics of measurement for fair evaluation of models and greater insights.

6.2.1 Quantitative Evaluation Parameters

The following metrics are numeric in nature and are generally used for objective evaluation of any machine learning model. Some of qualitative metrics that we will be using for the evaluation of neural networks are as follows:

1. **Mean Absolute Error (MAE)**: It is generally considered useful for regression related tasks and it uses Absolute difference. Formula:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{6.1}$$

2. **Root Mean Squared Error (RMSE)**: It is also a useful tool for regression related tasks and it uses Square root difference. Formula:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{6.2}$$

3. **Mean Squared Error (MSE)**: It is also considered useful for regression related tasks and it uses Squared difference. Formula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{6.3}$$

4. **Accuracy**: Its considered useful for classification based problems where the classes seems to be fairly balanced. Formula:

$$Accuracy = \frac{NumberOfCorrectPredictions}{TotalNumberOfPredictions} \tag{6.4}$$

5. **F1 Score**: Its considered useful for classification based problems where the classes seems to be imbalanced in nature. Formula:

$$F1Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{6.5}$$

6.2.2 Qualitative Evaluation Parameters

These metrics are different from the quantitative matrix as these are much more subjective in nature and are generally used to judge how a model is behaving. Some of the following that will be used for evaluation of neural networks are as follows:

1. **Interpretability**: Trying to understand how close is the prediction to the actual result of the mathematical problem.
2. **Generalization**: Trying to understand how is the model performing against the unseen data.
3. **Training Time**: The time it takes to complete the training of the model on a data-set.
4. **Inference Time**: The time it takes to complete the prediction of the model on some given data.
5. **Robustness**: To test how well the model performs with some induced noise.

6.3 Evaluation Matrix

By utilizing all the parameters discussed in [6.2] we propose two comprehensive evaluation matrix of ml models one with regards to mathematical problem and other with the types of mathematical problems any given neural network architecture is able to solve. For how we want to view these evaluations.These are as follows:

1. **Evaluation by Problem**[6.3.1]
2. **Evaluation by Architecture**[6.3.2]

These evaluation matrix consist of both qualitative and quantitative metrics of measurement for complete analysis.

For the sake of the thesis we will only implement **"Evaluation by Problem"** matrix as it provides the wider outlook on different types of architectures for a single problem.

At times even human evaluation could also be done to make sure that the models are running as expected and a fair evaluation is taking place.

6.3.1 Evaluation by Problem

In the following Evaluation matrix we take a specific Mathematical problem and evaluate different architectures against it.

Neural Network Architecture	FFNNs	CNNs	RNNs	GNNs	Transformers
Accuracy					
MSE					
MAE					
RMSE					
F1 Score					
Training Time					
Inference Time					
Interpretability					
Generalization					
Robustness					
Suitability for Specific Problems					

Table 6.1: Sample Evaluation matrix of a specific Mathematical problem.

6.3.2 Evaluation by Architecture

In the following Evaluation matrix we take a specific Neural Network Architecture and evaluate different Mathematical problem against it.

Mathematical Problems	Simple	Differentiation	Quadratic	Optimisation	Trigonometry
Accuracy					
MSE					
MAE					
RMSE					
F1 Score					
Training Time					
Inference Time					
Interpretability					
Generalization					
Robustness					
Suitability for NN Architecture					

Table 6.2: Sample Evaluation matrix of a specific Neural Network Architecture.

6.4 Actual Evaluations

6.4.1 Numerical problems

Addition dataset

Problem	Addition				
Architecture	**Perceptron**	**FFNN**	**CNN**	**RNN**	**LSTM**
Accuracy	High	High	High	Moderate	High
MSE	0.056	0.0007	1.551	0.0005	0.0016
MAE	0.23	0.027	0.003	0.023	0.012
RMSE	0.23	0.027	0.003	0.023	0.0406
F1 Score	High	High	High	Moderate	High
Training Time	22.66	32.38	46.75	29.30	32.07
Inference Time	0.11	0.140	0.158	0.11	0.128
Interpretability	High	Moderate	Low	Low	Low
Generalization	Low	Moderate	High	Moderate	High
Robustness	Low	Moderate	High	Moderate	High
Suitability	Moderate	High	High	High	High

Table 6.3: Evaluation matrix for Addition Dataset.

Subtraction dataset

Problem	Subtraction				
Architecture	**Perceptron**	**FFNN**	**CNN**	**RNN**	**LSTM**
Accuracy	High	High	High	High	High
MSE	0.0049	0.015	0.029	0.04	0.072
MAE	0.0055	0.107	0.003	0.048	0.233
RMSE	0.0064	0.12	0.004	0.021	0.268
F1 Score	High	High	High	Moderate	High
Training Time	21.00	32.98	21.28	27.55	31.92
Inference Time	0.11	0.12	0.11	0.11	0.1395
Interpretability	High	Moderate	Low	Low	Low
Generalization	Low	Moderate	High	Moderate	High
Robustness	Low	Moderate	High	Moderate	High
Suitability	Moderate	High	High	High	High

Table 6.4: Evaluation matrix for Subtraction Dataset.

Multiplication dataset

Problem	Multiplication Data				
Architecture	**Perceptron**	**FFNN**	**CNN**	**RNN**	**LSTM**
Accuracy	Low	Low	Low	Low	Low
MSE	High	High	High	High	High
MAE	High	High	High	High	High
RMSE	High	High	High	High	High
F1 Score	Low	Low	Low	Low	Low
Training Time	237.17	70.65	39.204	166.03	20.86
Inference Time	0.149	0.108	0.12	0.13	0.165
Interpretability	High	Moderate	Low	Low	Low
Generalization	Low	Moderate	High	Moderate	High
Robustness	Low	Moderate	High	Moderate	High
Suitability	Limited	Moderate	Limited	Not Ideal	Not Ideal

Table 6.5: Evaluation matrix for Multiplication Dataset.

Exponent dataset

Problem	Exponential Data				
Architecture	**Perceptron**	**FFNN**	**CNN**	**RNN**	**LSTM**
Accuracy	Low	Moderate	Moderate	Low	Moderate
MSE	High	High	High	High	High
MAE	High	High	High	High	High
RMSE	High	High	High	High	High
F1 Score	Low	Moderate	Moderate	Low	Moderate
Training Time	21.17	25.18	1.453	3.21	5.06
Inference Time	0.12	0.481	0.04	0.04	0.046
Interpretability	High	Moderate	Low	Low	Low
Generalization	Low	Moderate	High	Moderate	High
Robustness	Low	Moderate	High	Moderate	High
Suitability	Limited	Moderate	Not Ideal	Not Ideal	Not Ideal

Table 6.6: Evaluation matrix for Exponent Dataset.

Doubling dataset

Problem	Doubles the input value				
Architecture	**Perceptron**	**FFNN**	**CNN**	**RNN**	**LSTM**
Accuracy	High	High	High	Moderate	High
MSE	0.012	0.103	0.012	1.25	0.0002
MAE	0.1111	0.290	0.11	0.98	0.0059
RMSE	0.1111	0.321	0.11	1.12	0.014
F1 Score	High	High	High	Moderate	High
Training Time	21.083	33.28	27.42	27.51	28.55
Inference Time	0.100	0.155	0.118	0.11	0.123
Interpretability	High	Moderate	Low	Low	Low
Generalization	Low	Moderate	High	Moderate	High
Robustness	Low	Moderate	High	Moderate	High
Suitability	Moderate	High	High	Limited	High

Table 6.7: Evaluation matrix for Doubling Dataset.

Natural log dataset

Problem	Logarithmic Data				
Architecture	**Perceptron**	**FFNN**	**CNN**	**RNN**	**LSTM**
Accuracy	Low	Moderate	Moderate	Moderate	High
MSE	High	0.960	1.550	0.28	0.002
MAE	High	0.72	0.325	0.47	0.037
RMSE	High	0.979	0.406	0.53	0.048
F1 Score	Low	Moderate	Moderate	Moderate	High
Training Time	$\sim$	75.83	20.386	19.29	23.22
Inference Time	$\sim$	0.120	0.11	0.18	0.121
Interpretability	High	Moderate	Low	Low	Low
Generalization	Low	Moderate	High	Moderate	High
Robustness	Low	Moderate	High	Moderate	High
Suitability	Limited	Moderate	Not Ideal	Conditional	Conditional

Table 6.8: Evaluation matrix for Natural log Dataset.

Sine value dataset

Problem	Sine Values for a given input				
Architecture	**Perceptron**	**FFNN**	**CNN**	**RNN**	**LSTM**
Accuracy	Moderate	Moderate	Moderate	Moderate	Moderate
MSE	1.18	0.505	0.523	0.73	0.514
MAE	0.88	0.64	0.64	0.71	0.644
RMSE	1.08	0.710	0.72	0.85	0.717
F1 Score	Moderate	Moderate	Moderate	High	Moderate
Training Time	20.67	37.61	7.02	6.76	29.36
Inference Time	0.11	0.155	0.118	0.12	0.13
Interpretability	High	Moderate	Low	Low	Low
Generalization	Low	Moderate	High	Moderate	High
Robustness	Low	Moderate	High	Moderate	High
Suitability	Limited	High	Conditional	Conditional	Conditional

Table 6.9: Evaluation matrix for Sine value Dataset.

Cosine value dataset

Problem	Co-Sine Values for a given input				
Architecture	**Perceptron**	**FFNN**	**CNN**	**RNN**	**LSTM**
Accuracy	Moderate	Moderate	Moderate	Moderate	Moderate
MSE	$\sim$	0.495	0.674	0.69	0.495
MAE	$\sim$	0.634	0.690	0.70	0.634
RMSE	$\sim$	0.703	0.821	0.83	0.704
F1 Score	Moderate	Moderate	Moderate	Moderate	Moderate
Training Time	$\sim$	34.82	10.49	5.17	28.42
Inference Time	$\sim$	0.146	0.11	0.111	0.123
Interpretability	High	Moderate	Low	Low	Low
Generalization	Low	Moderate	High	Moderate	High
Robustness	Low	Moderate	High	Moderate	High
Suitability	Limited	High	Conditional	Conditional	Conditional

Table 6.10: Evaluation matrix for Cosine value Dataset.

Integration square and cosine value dataset

Problem	$[\int a^2cos(a) = b]$				
Architecture	**Perceptron**	**FFNN**	**CNN**	**RNN**	**LSTM**
Accuracy	Low	Moderate	High	High	$\sim$
MSE	High	3.47	0.487	0.134	$\sim$
MAE	High	1.38	0.21	0.175	$\sim$
RMSE	High	1.86	0.69	0.366	$\sim$
F1 Score	Low	Moderate	High	High	$\sim$
Training Time	$\sim$	236.8	56.68	32.02	$\sim$
Inference Time	$\sim$	0.12	0.11	0.14	$\sim$
Interpretability	High	Moderate	Low	Low	Low
Generalization	Low	Moderate	High	Moderate	High
Robustness	Low	Moderate	High	Moderate	High
Suitability	Not Ideal	Conditional	Conditional	Conditional	Conditional

Table 6.11: Evaluation matrix for Integration square and cosine value Dataset.

Integration square and sine value dataset

Problem	$[\int a^2sin(a) = b]$				
Architecture	**Perceptron**	**FFNN**	**CNN**	**RNN**	**LSTM**
Accuracy	Low	Moderate	High	High	$\sim$
MSE	High	10.15	0.147	0.057	$\sim$
MAE	High	2.28	0.275	0.15	$\sim$
RMSE	High	3.18	0.38	0.239	$\sim$
F1 Score	Low	Moderate	High	High	$\sim$
Training Time	$\sim$	48.219	32.77	18.01	$\sim$
Inference Time	$\sim$	0.17	0.115	0.130	$\sim$
Interpretability	High	Moderate	Low	Low	Low
Generalization	Low	Moderate	High	Moderate	High
Robustness	Low	Moderate	High	Moderate	High
Suitability	Not Ideal	Conditional	Conditional	Conditional	Conditional

Table 6.12: Evaluation matrix for Integration square and sine value Dataset.

Integration square and tan value dataset

Problem	$[\int a^2 tan(a) = b]$				
Architecture	**Perceptron**	**FFNN**	**CNN**	**RNN**	**LSTM**
Accuracy	Low	Low	Low	Low	Low
MSE	High	High	High	High	High
MAE	High	High	High	High	High
RMSE	High	High	High	High	High
F1 Score	Low	Low	Low	Low	Low
Training Time	∼	235.31	42.68	5.551	∼
Inference Time	∼	0.12	0.35	0.1188	∼
Interpretability	High	Moderate	Low	Low	Low
Generalization	Low	Moderate	High	Moderate	High
Robustness	Low	Moderate	High	Moderate	High
Suitability	Not Ideal	Conditional	Conditional	Conditional	Conditional

Table 6.13: Evaluation matrix for Integration square and tan value Dataset.

6.4.2 Probability Distribution Problems

Normal distribution dataset

Problem	Normal Probabilistic distribution Values				
Architecture	**Perceptron**	**FFNN**	**CNN**	**RNN**	**LSTM**
Accuracy	Low	Moderate	Moderate	Moderate	Moderate
MSE	High	0.250	0.250	0.25	0.25
MAE	High	0.50	0.50	0.500	0.500
RMSE	High	0.50	0.50	0.500	0.500
F1 Score	Low	Moderate	Moderate	Moderate	Moderate
Training Time	∼	25.41	13.0	28.28	∼
Inference Time	∼	0.108	0.11	0.113	∼
Interpretability	High	Moderate	Low	Low	Low
Generalization	Low	Moderate	High	Moderate	High
Robustness	Low	Moderate	High	Moderate	High
Suitability	Not Ideal	Conditional	Conditional	Conditional	Conditional

Table 6.14: Evaluation matrix for Normal distribution Dataset.

Binomial distribution dataset

Problem	Binomial Probabilistic distribution Values				
Architecture	**Perceptron**	**FFNN**	**CNN**	**RNN**	**LSTM**
Accuracy	Low	Moderate	Moderate	Moderate	Moderate
MSE	High	0.250	0.250	0.25	0.25
MAE	High	0.50	0.50	0.500	0.500
RMSE	High	0.50	0.50	0.500	0.500
F1 Score	Low	Moderate	Moderate	Moderate	Moderate
Training Time	12.9	26.24	13.10	27.11	27.15
Inference Time	0.112	0.22	0.10	0.111	0.15
Interpretability	High	Moderate	Low	Low	Low
Generalization	Low	Moderate	High	Moderate	High
Robustness	Low	Moderate	High	Moderate	High
Suitability	Not Ideal	Conditional	Conditional	Conditional	Conditional

Table 6.15: Evaluation matrix for Binomial distribution Dataset.

Gamma distribution dataset

Problem	Gamma Probabilistic distribution Values				
Architecture	**Perceptron**	**FFNN**	**CNN**	**RNN**	**LSTM**
Accuracy	Low	Moderate	Moderate	Moderate	Moderate
MSE	High	0.250	0.250	0.25	0.25
MAE	High	0.50	0.50	0.500	0.500
RMSE	High	0.50	0.50	0.500	0.500
F1 Score	Low	Moderate	Moderate	Moderate	Moderate
Training Time	$\sim$	24.53	17.12	4.897	$\sim$
Inference Time	$\sim$	0.11	0.12	0.1182	$\sim$
Interpretability	High	Moderate	Low	Low	Low
Generalization	Low	Moderate	High	Moderate	High
Robustness	Low	Moderate	High	Moderate	High
Suitability	Not Ideal	Conditional	Conditional	Conditional	Conditional

Table 6.16: Evaluation matrix for Gamma distribution Dataset.

Poisson distribution dataset

Problem	Poisson Probabilistic distribution Values				
Architecture	**Perceptron**	**FFNN**	**CNN**	**RNN**	**LSTM**
Accuracy	Low	Moderate	Moderate	Moderate	Moderate
MSE	High	0.250	0.250	0.25	0.25
MAE	High	0.50	0.50	0.500	0.500
RMSE	High	0.50	0.50	0.500	0.500
F1 Score	Low	Moderate	Moderate	Moderate	Moderate
Training Time	∼	24.90	8.311	5.331	∼
Inference Time	∼	0.115	0.10	0.118	∼
Interpretability	High	Moderate	Low	Low	Low
Generalization	Low	Moderate	High	Moderate	High
Robustness	Low	Moderate	High	Moderate	High
Suitability	Not Ideal	Conditional	Conditional	Conditional	Conditional

Table 6.17: Evaluation matrix for Poisson distribution Dataset.

Exponential distribution dataset

Problem	Exponential Probabilistic distribution Values				
Architecture	**Perceptron**	**FFNN**	**CNN**	**RNN**	**LSTM**
Accuracy	Low	Moderate	Moderate	Moderate	Moderate
MSE	High	0.250	0.250	0.25	0.25
MAE	High	0.50	0.50	0.500	0.500
RMSE	High	0.50	0.50	0.500	0.500
F1 Score	Low	Moderate	Moderate	Moderate	Moderate
Training Time	∼	23.52	13.12	25.54	∼
Inference Time	∼	0.17	0.113	0.117	∼
Interpretability	High	Moderate	Low	Low	Low
Generalization	Low	Moderate	High	Moderate	High
Robustness	Low	Moderate	High	Moderate	High
Suitability	Not Ideal	Conditional	Conditional	Conditional	Conditional

Table 6.18: Evaluation matrix for Exponential distribution Dataset.

6.4.3 Special dataset

CIFAR10

Problem	CIFAR10 image dataset				
Architecture	**AlexNet**	**VGG16**	**ResNet**	**ConvNext**	**CBAM**
Accuracy	High	High	High	High	High
Suitability	Ideal	Ideal	Ideal	Ideal	Ideal

Table 6.19: Evaluation matrix for CIFAR10.

IMDb Sentiment Analysis

Problem	IMDb Movie review				
Architecture	**Transformer**	**FFNN**	**CNN**	**RNN**	**LSTM**
Accuracy	High	Moderate	Low	High	High
Suitability	Ideal	Conditional	Not Ideal	High	Ideal

Table 6.20: Evaluation matrix for IMDb Sentiment Analysis.

Chapter 7

Conclusion and Future Work

7.1 Introduction

In the following section we will discuss the evaluations of various implemented neural network architectures models while also concluding the thesis, making recommendation and discussing the future work.

7.2 Conclusion

Through this thesis project, we explored[2] benefits, common weaknesses and usefulness of various machine learning models ranging from simple to some state-of-the-art neural network architectures with respect to various mathematical problems and common generic data sets.

Due to implementation[5.3] of so many machine learning models this study aims to increase awareness and suggest usefulness of these distinct neural network architectures for different kinds of problems or tasks they might be facing.

Evaluation[6] of these neural network architectures followed a comprehensive set of popular parameters against which all these machine learning models were tested. These evaluations helped us in gaining insight of strengths and weaknesses of these distinct neural network architectures.

To conclude this dissertation, the following project can work as a building block for future research and development in the neural network architectures itself. As this provides valuable insights on the kind of problems these architectures are good with and where these machine learning models suffer.

As new architectures continue to emerge this research can be further expanded to include all of them to develop new more complex and deeper neural networks. As development of these models will help us to create more high performance and versatile tools Which help us in fully realising the power of Neural networks.

7.3 Architecture Recommendation

During the evaluation of various neural networks, we discovered that even considerably basic architectures like feed-forward neural networks[2.4.2] and convolutional neural networks demonstrated a very promising results When they were trained on controlled synthetic mathematical data.

We even saw extremely high accuracies in some of the mathematical problems while also maintaining an incredibly low count on the error parameters.

In terms of speed perceptron[2.4.1] was the fastest as it was the simplest model but in terms of multiple layers, we still saw feed forward neural networks as the fastest amongst all the distinct kinds of neural network models.

But the parameter of interpretability is where simple machine learning models like perceptron and feed-forward neural network face a huge challenge as explaining ”How they reached the final solution?” becomes extremely tedious which only worsens interpretability in case of more complex problems.

On the other hand, models which required sequential data like recurrent neural networks[2.4.4] and long short-term memory based RNN's displayed relatively low performance when tested on controlled synthetic data sets. This suggests that these architectures might not be a good fit for mathematical problems.
Which also suggests that solving a basic mathematical problem might Require a lot of parallel computations then originally led on.

The surprise superstars of all these neural network architectures in relationship to mathematical problems comes out as convolutional neural networks[2.4.3].
As the following neural network models while not originally made for the purpose of solving mathematical problems they still achieved a good enough accuracy and low on error parameters for many problems.
These models might not be as fast as feed forward neural networks, but these models still displayed a respectable amount of performance on the various kinds of mathematical problems.

While many advanced state-of-the-art and neural network architectures[5.3] displayed overall higher performance on the Established generic data sets.
For example, models like ConvNext, VGG16, and LSTMs achieved extremely high accuracy on the image classification dataset while models like Transformers and LSTMs also displayed high accuracy on text classification data set.
While these models displayed overall high performance, but it quickly became apparent that they are very hard to train because of their highly complex design large number of layers.

7.4 Future Work

The following thesis can be further explored with a lot of different avenues.
Some of them are as follows:

1. **New architectures**: As the number of neural network architectures increases the scope of this project increases with it.
2. **More complex Problems**: We can also explore inclusion of more complex mathematical problems for evaluation of these modals.
3. **Real-world data**: This project will really benefit from integration of real-world data. As this will allow these machine learning model to function and be evaluated in real world scenarios.
4. **XAI**: This project can also benefit from a very novel but impressive concept off explainable AI(XAI) as these explainable AI models make it quite easy to interpret the decision making of these neural network models removing the black box aspect of these architectures.
5. **Heterogeneous architectures**: Though not much explored in the following thesis we can also investigate the impact of heterogeneous architectures. As this will allow these machine learning models in overcoming their common shortcomings.
6. **Multi-modal architectures**: We can also explorer some neural network architectures that are multi-modal in nature. As these kinds of machine learning models can handle different kinds of data. Hence, they might be very successful in handling mathematical problem data too.

Chapter 8

Project Management

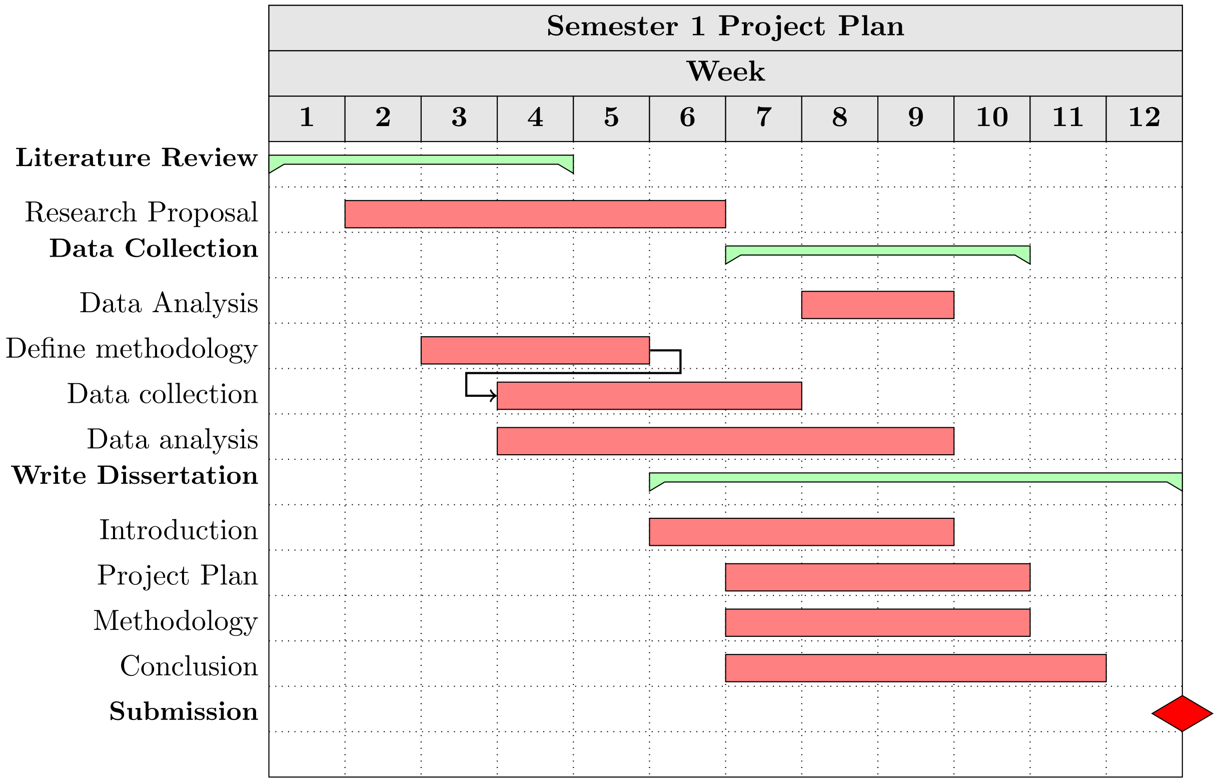
8.1 Introduction

In the following section of the report, we will discuss about the implementation timeline, literature review timeline and various other social aspects of the project planning for the thesis.

8.2 Project Planning

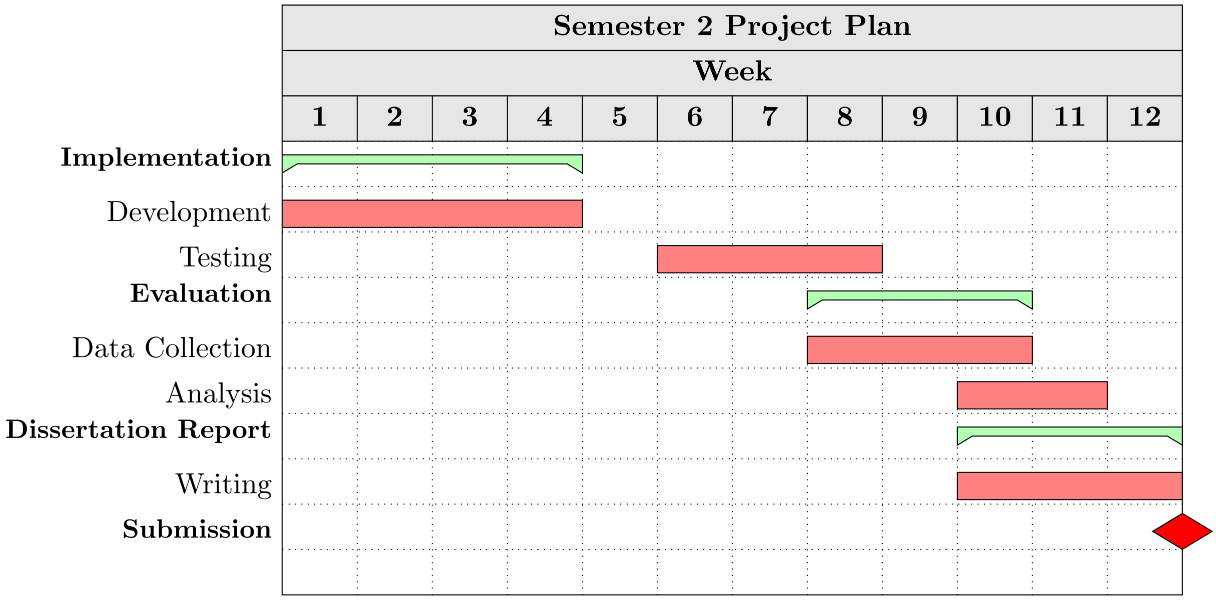
Semester 1:
In the first semester the main goal is to finalise the research topic and finding out weather topic itself is feasible for implementation or not. Gantt Chart below shows the timeline of literature review, research proposal, data analysis, project planning, writing dissertation and report submission.

Gantt Chart Semester 1



Semester 2:
In the second semester after we have completed the literature review the main aim is to implement what we have committed in literature review. Hence, this semester focuses on implementation development evaluation testing and writing the final dissertation report before submission.

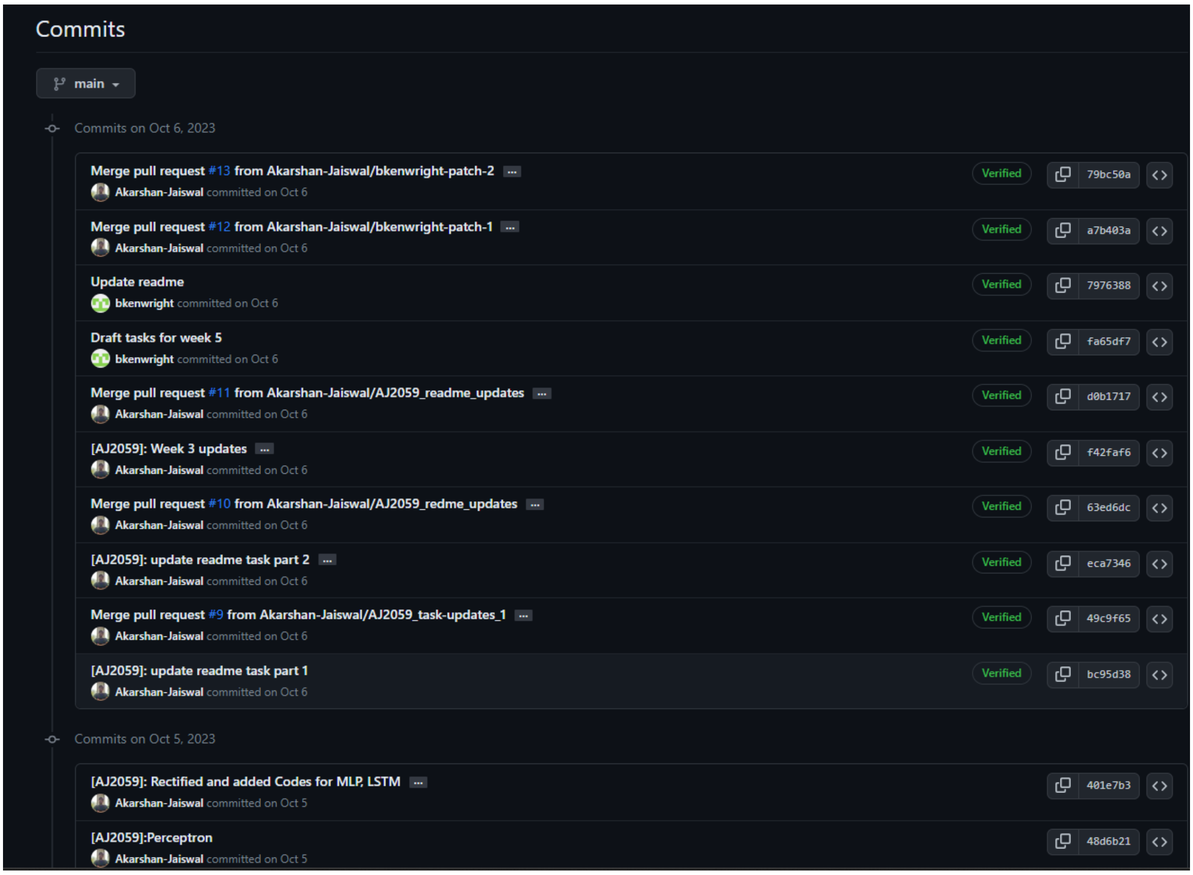
Gantt Chart Semester 2



Appendix A

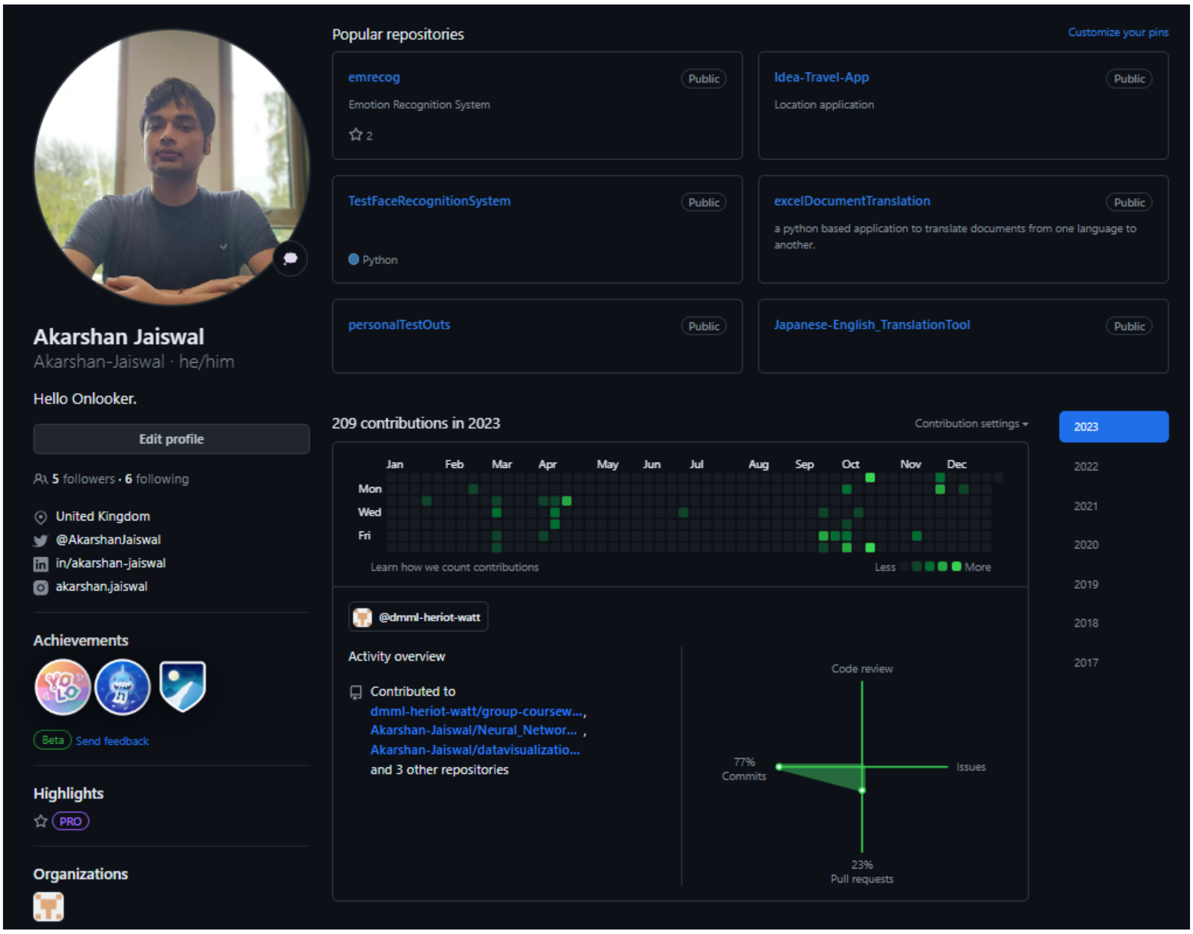
A.1 Journal/Project Management - GitHub Repository

Project planning is core aspect of any dissertation (especially for the initial groundwork). Key tool is Git/GitHub



```
Commits

main

Commits on Oct 6, 2023

Merge pull request #13 from Akarshan-Jaiswal/bkenwright-patch-2 ...       Verified   79bc50a
Akarshan-Jaiswal committed on Oct 6

Merge pull request #12 from Akarshan-Jaiswal/bkenwright-patch-1 ...       Verified   a7b403a
Akarshan-Jaiswal committed on Oct 6

Update readme                                                             Verified   7976388
bkenwright committed on Oct 6

Draft tasks for week 5                                                    Verified   fa65df7
bkenwright committed on Oct 6

Merge pull request #11 from Akarshan-Jaiswal/AJ2059_readme_updates ...    Verified   d0b1717
Akarshan-Jaiswal committed on Oct 6

[AJ2059]: Week 3 updates ...                                              Verified   f42faf6
Akarshan-Jaiswal committed on Oct 6

Merge pull request #10 from Akarshan-Jaiswal/AJ2059_redme_updates ...     Verified   63ed6dc
Akarshan-Jaiswal committed on Oct 6

[AJ2059]: update readme task part 2 ...                                   Verified   eca7346
Akarshan-Jaiswal committed on Oct 6

Merge pull request #9 from Akarshan-Jaiswal/AJ2059_task-updates_1 ...     Verified   49c9f65
Akarshan-Jaiswal committed on Oct 6

[AJ2059]: update readme task part 1                                       Verified   bc95d38
Akarshan-Jaiswal committed on Oct 6

Commits on Oct 5, 2023

[AJ2059]: Rectified and added Codes for MLP, LSTM ...                                401e7b3
Akarshan-Jaiswal committed on Oct 5

[AJ2059]:Perceptron                                                                  48d6b21
Akarshan-Jaiswal committed on Oct 5
```

Figure A.1: Evidence of past activities over the initial phases of development.



Akarshan Jaiswal
Akarshan-Jaiswal · he/him

Hello Onlooker.

Edit profile

5 followers · 6 following

United Kingdom
@AkarshanJaiswal
in/akarshan-jaiswal
akarshan.jaiswal

Achievements

Beta Send feedback

Highlights

☆ PRO

Organizations

Popular repositories — Customize your pins

Repository	Visibility	Description	Details
emrecog	Public	Emotion Recognition System	☆ 2
Idea-Travel-App	Public	Location application	
TestFaceRecognitionSystem	Public		● Python
excelDocumentTranslation	Public	a python based application to translate documents from one language to another.	
personalTestOuts	Public		
Japanese-English_TranslationTool	Public		

209 contributions in 2023 — Contribution settings ▾

2023
2022
2021
2020
2019
2018
2017

Learn how we count contributions

Less 🟩🟩🟩🟩🟩 More

@dmml-heriot-watt

Activity overview

Contributed to
dmml-heriot-watt/group-coursew...,
Akarshan-Jaiswal/Neural_Networ... ,
Akarshan-Jaiswal/datavisualizatio...
and 3 other repositories

Figure A.2: Additional features/explanations/evidence from GitHub.

.

A.2 D1 Submission

Marking criteria for the D1 Submission is shown in Table A.1.

The appendix also features supplementary information, including additional points and comments, aimed at offering context and guidance to markers during the evaluation process. This approach is particularly valuable because it allows me to consider specific aspects of the project, such as the Journal and Project Tools, which may not be immediately evident from the main assessment rubric. This ensures that the assessment process is thorough and fair, benefiting both students and markers. The inclusion of this comprehensive marking criteria in the appendix underscores my commitment to fair and constructive evaluation, empowering students to understand how their work is assessed and enabling markers to provide meaningful feedback to support continuous improvement (see **"How we mark your project"** on CANVAS for Details).

Criteria	General Guidelines/Checklist
Motivation and scope (20%)	• Is the topic meaningful, complex and challenging? • Are the objectives and aims clearly stated? • Are the main areas of investigation for the project identified? • Has the student provided ideas and approaches of original thinking? • Is there evidence of a clear understanding of the project area/research topic? • Is there any novelty in the work/ is the work a contribution to the area?
Project background (20%)	• Is there evidence of sufficient background reading? • Is there evidence of a clear understanding of the project area/research topic? • Is the literature study up to date and critically evaluated? • How pertinent is the background/referenced material? • Has the student used appropriate tools/software? • Have the appropriate design methodologies been employed? • Are the appendices relevant?
Implementation/work conducted (20%)	• Does the student appear to have undertaken a significant volume of work? • Was the student required to master new material to enable them to undertake the project? • Is the topic investigated to an appropriate depth? • Does the dissertation show a deep understanding of the topic?
Evaluation and conclusions (20%)	• Are the results presented clearly in a logical manner? • Are problems and difficulties explained? • Does the student demonstrate an understanding and interpretation of results and their significance? • Is the application/product complex? Or is it of limited functionality? • Does the student demonstrate an appropriate level of understanding of the complexities of the project? • Is there any critical evaluation of the project? • Has the student suggested future work? • Are evaluation and recommendations coherent and logical?
Quality of document (20%)	• Is the writing clear, concise and with good English? • Is the dissertation sensibly structured into chapters and sections? • Is the dissertation of an appropriate length? • Is the approach adopted well justified? • How well did the student discuss and explain their own work? • Is the dissertation as a document of a high standard, appropriate for a Masters degree?

Table A.1: D1 Marking Rubric.

Student Declaration of Authorship

Course code and name:	Dissertation F21MP
Type of assessment:	~~**Group**~~ **/ Individual** *(delete as appropriate)*
Coursework Title:	Dissertation Report
Student Name:	Akarshan Jaiswal
Student ID Number:	H00425264

Declaration of authorship. By signing this form:

- **I declare** that the work I have submitted for individual assessment OR the work I have contributed to a group assessment, is entirely my own. I have NOT taken the ideas, writings or inventions of another person and used these as if they were my own. My submission or my contribution to a group submission is expressed in my own words. Any uses made within this work of the ideas, writings or inventions of others, or of any existing sources of information (books, journals, websites, etc.) are properly acknowledged and listed in the references and/or acknowledgements section.
- I confirm that I have read, understood and followed the University's Regulations on plagiarism as published on the University's website, and that I am aware of the penalties that I will face should I not adhere to the University Regulations.
- I confirm that I have read, understood and avoided the different types of plagiarism explained in the University guidance on Academic Integrity and Plagiarism

Student Signature *(type your name):* Akarshan Jaiswal

Date: April 17, 2024

Copy this page and insert it into your coursework file in front of your title page.
For group assessment each group member must sign a separate form and all forms must be included with the group submission.

Your work will not be marked if a signed copy of this form is not included with your submission.

Bibliography

[Aggarwal, 2018] Aggarwal, C. C. (2018). An introduction to neural networks. *Neural Networks and Deep Learning*, page 1–52. 4

[Alzubaidi et al., 2021] Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., and Farhan, L. (2021). Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. *Journal of Big Data, URL: https://doi.org/10.1186/s40537-021-00444-8, Accessed:10/12/2023*. 1

[Bianco et al., 2018] Bianco, S., Cadene, R., Celona, L., and Napoletano, P. (2018). Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6:64270–64277. 1

[Bishop, 2007] Bishop, C. M. (2007). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition. 8, 18

[Cao et al., 2021] Cao, Y., Sun, Z., and Sartoretti, G. (2021). Dan: Decentralized attention-based neural network to solve the minmax multiple traveling salesman problem. *arXiv preprint arXiv:2109.04205*. 13

[Caterini, 2017] Caterini, A. (2017). A novel mathematical framework for the analysis of neural networks. Master's thesis, University of Waterloo. 9, 19

[Caudill, 1989] Caudill, M. (1989). A basic introduction to neural networks. *UNIVERSITY OF WISCONSIN–MADISON, URL: https://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html*. 4

[Chandola et al., 2009] Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM Comput. Surv.*, 41. 8, 19

[Charton and Lample, 2022] Charton, F. and Lample, G. (2022). Using neural networks to solve advanced mathematics equations. *AI.Meta Blog, URL: https://ai.meta.com/blog/using-neural-networks-to-solve-advanced-mathematics-equations/*. 9

[Cho et al., 2014] Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*. 9, 10, 19

[Chowdhury, 2023] Chowdhury, P. (2023). Perceptron: A simple yet mighty machine learning algorithm. *Medium, URL: https://medium.com/@cprasenjit32/perceptron-a-simple-yet-mighty-machine-learning-algorithm-9ff6b7d86a71.* vii, 6

[Chung et al., 2015] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2015). Gated feedback recurrent neural networks. In *International conference on machine learning*, pages 2067–2075. PMLR. 10, 19

[Cohen and Giryes, 2023] Cohen, G. and Giryes, R. (2023). Generative adversarial networks. In *Machine Learning for Data Science Handbook: Data Mining and Knowledge Discovery Handbook*, pages 375–400. Springer. 12

[Cong and Zhou, 2023] Cong, S. and Zhou, Y. (2023). A review of convolutional neural network architectures and their optimizations. *Artificial Intelligence Review, URL: https://doi.org/10.1007/s10462-022-10213-5, Accessed:10/12/2023*, 56. 1

[Cordoni, 2020] Cordoni, F. (2020). A comparison of modern deep neural network architectures for energy spot price forecasting. *Digital Finance, URL: https://doi.org/10.1007/s42521-020-00022-2, Accessed:10/12/2023*, 2. 1

[Ericson and Jensfelt, 2022] Ericson, L. and Jensfelt, P. (2022). Floorgent: Generative vector graphic model of floor plans for robotics. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12485–12491. IEEE. 13

[fdeloche, 2017] fdeloche (2017). Recurrent neural network unfold.svg. vii, 9

[Ghorakavi, 2018] Ghorakavi, V. (2018). Neural networks — a beginners guide. *Geeks for geeks, URL: https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/.* 4

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, 27. 11, 12

[Goodfellow et al., 2016] Goodfellow, I. J., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press, Cambridge, MA, USA. http://www.deeplearningbook.org. 8, 18

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778. 16, 35

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780. 10, 19

[Karagiannakos, 2020] Karagiannakos, S. (2020). Graph neural networks - an overview. *AI Summer*. vii, 15

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25. 16, 35

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436. 8, 18

[Ledig et al., 2017] Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., et al. (2017). Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690. 12

[Lei et al., 2022] Lei, K., Guo, P., Wang, Y., Wu, X., and Zhao, W. (2022). Solve routing problems with a residual edge-graph attention neural network. *Neurocomputing*, 508:79–98. 15

[Liu and Xu, 2023] Liu, Z. and Xu, F. (2023). Interpretable neural networks: principles and applications. *Frontiers in Artificial Intelligence*, 6. 15

[Liu and Zhou, 2022] Liu, Z. and Zhou, J. (2022). *Introduction to graph neural networks*. Springer Nature. 14, 15

[Ma et al., 2019] Ma, X., Spinner, S., Venditti, A., Li, Z., and Tang, S. (2019). Initial margin simulation with deep learning. *SSRN Electronic Journal*. vii, 7

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics, URL: https://doi.org/10.1007/BF02478259, Accessed:10/12/2023*, 5. 5

[Minsky and Papert, 1969] Minsky, M. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA. 5

[Navaneetha Krishnan et al., 2023] Navaneetha Krishnan, M., Swetha, G., Saravanakumar, R., Nandhinishri, S., and Arthi, R. (2023). Comparative analysis of convolutional neural network and character recognition techniques for handwritten mathematical equation solver. *Journal of Survey in Fisheries Sciences*, 10(4S):1609–1632. 9

[Paliwal et al., 2020] Paliwal, A., Loos, S., Rabe, M., Bansal, K., and Szegedy, C. (2020). Graph representations for higher-order logic and theorem proving. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2967–2974. 15

[Randle et al., 2020] Randle, D., Protopapas, P., and Sondak, D. (2020). Unsupervised learning of solutions to differential equations with generative adversarial networks. *arXiv preprint arXiv:2007.11133*. 12

[Ren et al., 2018] Ren, S., Chen, G., Li, T., Chen, Q., and Li, S. (2018). A deep learning-based computational algorithm for identifying damage load condition: An artificial intelligence inverse problem solution for failure analysis. *Computer Modeling in Engineering &amp;amp; Sciences*, 117(3):287–307. vii, 3

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review, URL: https://doi.org/10.1037/h0042519, Accessed:10/12/2023*, 65. 5, 7, 18

[Sanjeevi, 2018] Sanjeevi, M. (2018). Chapter 10: Deepnlp - recurrent neural networks with math. 10, 19

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*. 16, 35

[Sit et al., 2020] Sit, M., Demiray, B., Xiang, Z., Ewing, G., Sermet, Y., and Demir, I. (2020). A comprehensive review of deep learning applications in hydrology and water resources. *Water Science and Technology*, 82. vii, 8

[Valkov, 2018] Valkov, V. (2018). Making a predictive keyboard using recurrent neural networks-tensorflow for hackers, part v. vii, 9

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30. 13

[Wang et al., 2019] Wang, Q., Liu, M., Zhang, W., Guo, Y., and Li, T. (2019). Automatic proofreading in chinese: detect and correct spelling errors in character-level with deep neural networks. In *Natural Language Processing and Chinese Computing: 8th CCF International Conference, NLPCC 2019, Dunhuang, China, October 9–14, 2019, Proceedings, Part II 8*, pages 349–359. Springer. vii, 13

[Werbos, 1988] Werbos, P. J. (1988). Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356. 5

[Woo et al., 2018] Woo, S., Park, J., Lee, J.-Y., and Kweon, I. S. (2018). Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19. 16, 35

[Yalçın, 2023] Yalçın, O. G. (2023). Image generation in 10 minutes with generative adversarial networks. *Medium*. vii, 11

[Ye et al., 2019] Ye, Z., Gilman, A., Peng, Q., Levick, K., Cosman, P., and Milstein, L. (2019). Comparison of neural network architectures for spectrum sensing. In *2019 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. 1

[Younesi et al., 2024] Younesi, A., Ansari, M., Fazli, M., Ejlali, A., Shafique, M., and Henkel, J. (2024). A comprehensive survey of convolutions in deep learning: Applications, challenges, and future trends. *arXiv preprint arXiv:2402.15490*. 16, 35

[Zhang, 2003] Zhang, P. (2003). Zhang, g.p.: Time series forecasting using a hybrid arima and neural network model. neurocomputing 50, 159-175. *Neurocomputing*, 50:159–175. 8, 19

[Zhou et al., 2020] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI open*, 1:57–81. 14