L-Università ta' Malta
Faculty of Science

Department of
Statistics &
Operations Research

Seismic Source Characterization in the Central Mediterranean Region

Giulia Maria Tabone

Oct 2025

Supervisor: Dr Fiona Sammut

Co-Supervisors: Prof. David Suda, Dr Matthew Agius

A dissertation presented to the Faculty of Science in partial fulfilment of the requirements for the degree of Master of Science at the University of Malta

Abstract

Giulia Maria Tabone, M.Sc.
Department of Statistics & Operations Research
University of Malta
Oct 2025

The process of *earthquake source characterization* involves determining the key properties of an earthquake and its origin, including its *location*, as indicated by *latitude and longitude coordinates*, its *depth* and its *magnitude*. This dissertation focuses on establishing a statistical model, by combining different Neural Network (NN) architectures including the Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN) and Graph Neural Network (GNN), in order to characterize the source of the earthquakes occurring in the central Mediterranean region, particularly concentrating on the Maltese islands, Sicily and the surrounding areas. The models considered are trained and validated on earthquake data recorded between 2013 and 2024. These data were obtained from seismic stations positioned around the Maltese Islands and Sicily, which are installed and maintained by the Istituto Nazionale di Geofisica e Vulcanologia (INGV) and the University of Malta's Seismic Monitoring and Research Group (SMRG). The objective is to predict earthquake source parameters using station coordinates and waveform features. Each earthquake is represented as a graph in which the vertices correspond to stations, and the associated features are assigned to these nodes. For each event, the model outputs latitude, longitude, depth, and magnitude. Two model architectures are considered, namely, an *edgeless graph model* and a *dynamic edges GNN*. To identify the optimal model of each architecture, a systematic series of experiments, including hyperparameter tuning, regularization techniques, restricting the analysis to a more localized region and ensemble modelling, are conducted. The best two models are then fit on test data comprising events from January 2025 to August 2025. The edgeless graph architecture emerged as the best-performing architecture.

Acknowledgements

The completion of this dissertation was made possible through the dedication and unwavering support of a number of truly remarkable individuals.

Firstly, I would like to express my deepest gratitude to my supervisor, Dr. Fiona Sammut, for her invaluable guidance and encouragement throughout this journey. I would also like to extend my sincere thanks to my co-supervisor, Prof. David Paul Suda, for his additional feedback and guidance on my dissertation and project-based components of my Master's, and to my co-supervisor, Dr. Matthew Agius, for his insightful expertise in seismology and constructive comments on my work. I am further grateful to Dr. Monique Borg Inguanez for her patience and helpful feedback throughout the project-based components of my Master's.

Heartfelt appreciation also goes to Hayley for her constant love, support, and patience, for standing by me through both the best and most challenging times, and for always finding ways to make life easier. I would also like to thank my family and friends for their encouragement and understanding.

Lastly, I am grateful to my employers and colleagues at the Central Bank of Malta for their support during my studies. In particular, I would like to thank Lara for always being there to listen, offering advice, and keeping me grounded with her book recommendations.

Contents

List of Figures

List of Tables

List of Abbreviations

SSH Sea Surface wave Height

RMS Root Mean Square

SNR Signal-to-Noise Ratio

KNN k-Nearest Neighbour

NN Neural Network

RHS Right Hand Side

ANN Artificial Neural Network

INGV Istituto Nazionale di Geofisica e Vulcanologia

SMRG Seismic Monitoring and Research Group

ReLU Rectified Linear Unit

UAT Universal Approximation Theorem

MSE Mean Squared Error

NRMSE Normalized Root Mean Squared Error

MAE Mean Absolute Error

SGD Stochastic Gradient Descent

GNN Graph Neural Network

MLP Multi-Layer Perceptron

CNN Convolutional Neural Network

GMP Global Max Pooling

FCN Fully Convolutional Network

STGNN Spatiotemporal Graph Neural Network

MSEP Mean Squared Error of Prediction

MAEP Mean Absolute Error of Prediction

NRMSEP Normalized Root Mean Squared Error of Prediction

NN4G Neural Network for Graphs

NLP Natural Language Processing

FFT Fast Fourier Transform

DFT Discrete Fourier Transform

WCV Within Cluster Variation

MPNN Message-Passing Neural Network

WL Weisfeiler-Leman

List of Notation

$\boldsymbol{x}$	n-vector of inputs
$\boldsymbol{w}$	n-vector of weights
w_0	bias
$\sigma(.)$	activation function
y	perceptron output
z	output of neuron prior to the application of the activation function
$\hat{y}$	NN predicted output
$\mathbb{X}$	$(N \times n_x)$-matrix of inputs
$\boldsymbol{h}$	hidden layer containing n_h units
$\mathbb{W}^{x,h}$	$(n_x \times n_h)$-weight matrix containing the weights joining the input layer to the hidden layer
$\boldsymbol{w}^{h,y}$	n_h-weight vector containing the weights joining the hidden layer to the output layer
$\boldsymbol{w}_0^h$	n_h-bias vector associated with the hidden layer
w_0^y	n_o-bias term associated with the output
$J(\hat{\boldsymbol{y}}, \boldsymbol{y})$	loss function
t	denotes the iteration number
α_t	learning rate at the t^{th} iteration
$w_{ii'}^{(l,l+1)}$	weight joining the i^{th} neuron in layer l to the i'^{th} in layer $(l+1)$
λ	regularization parameter
L	final layer of a neural network
n_l	number of units in layer l

$\mathbb{I}$	input tensor of size $(N_s \times N_t \times D)$
D	number of seismic components
$\mathbb{K}$	filter of size $(K_{N_s} \times K_{N_t} \times D)$
$\mathbb{Z}(i, j, d)$	(i, j, d)-position on feature map
$\mathbb{O}(i, j, d)$	(i, j, d)-position on activation map
$\mathbb{P}(i, j)$	pooling layer output at position (i, j)
$\boldsymbol{V}$	q-vector obtained after flattening pooling layer output
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	graph where $\mathcal{V}$ is a set of entities, called nodes (or vertices), and $\mathcal{E}$ is a set of relationships, called edges, between these nodes.
$\mathbb{A}$	adjacency matrix
$\mathcal{G} = (\mathcal{V}, \mathbb{A}, \mathbb{X})$	attributed graph
$\lvert\mathcal{V}\rvert$	number of nodes
$\boldsymbol{x}_u$	feature vector of a node $u \in \mathcal{V}$
$\mathcal{N}(u)$	graph neighbourhood of a node $u \in \mathcal{V}$
$\boldsymbol{h}_u$	hidden embedding of a node $u \in \mathcal{V}$
$\boldsymbol{m}_{\mathcal{N}(u)}$	the "message" that is aggregated from node u's graph neighbourhood $\mathcal{N}(u)$
$\boldsymbol{z}_u$	node embedding of a node $u \in \mathcal{V}$

Chapter 1

An Overview of Earthquake Source Characterization

1.1 Introduction

The process of *earthquake source characterization* involves predicting the key properties of an earthquake at its origin, including its *location*, as indicated by by latitude and longitude coordinates, its *depth*, and its *magnitude*. As will be seen in Section 1.3, characterization of earthquake source parameters plays a crucial role in many seismic applications, such as earthquake early warning, hazard assessment, and emergency response (L. Li et al., 2020).

In this dissertation, the term *prediction* is used in the statistical and machine-learning sense, referring to the inference of unknown earthquake source parameters from observed seismic data following event detection, rather than the forecasting of earthquakes prior to their occurrence. Recent research has demonstrated that earthquake source parameters, including latitude, longitude, depth, and magnitude, can be predicted by combining information from seismic station geometry and recorded waveform data using data-driven and machine-learning-based approaches (Van den Ende & Ampuero, 2020; X. Zhang et al., 2022).

A seismic waveform is a record of ground motion over time collected by a sensor and digitalised at regular intervals, illustrating how the Earth's surface moves in reaction to passing seismic waves. Each seismic station records three such waveforms, referred to as components, aligned along orthogonal axes: *Vertical (Z)*, *North-South (N)* and *East-West (E)*. An example of a three-component (orthogonal) waveform is shown in Figure 1.1.1.

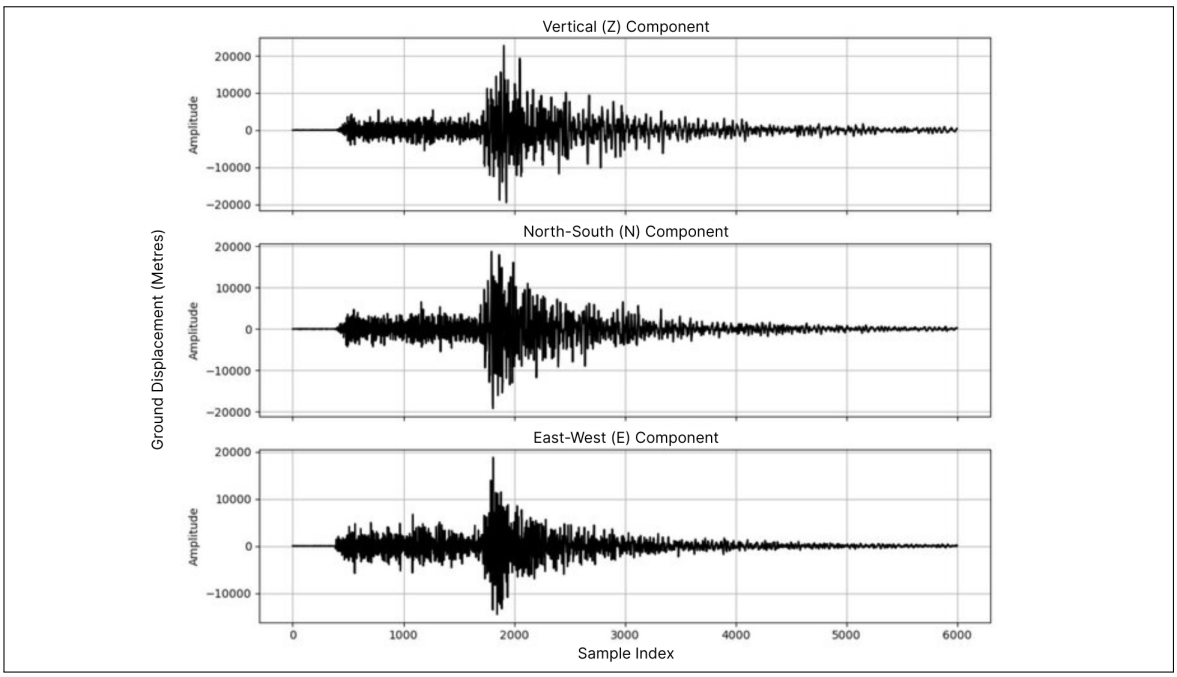

Figure 1.1.1: An example of an orthogonal 3-component waveform (Sharma, 2024).

With reference to Figure 1.1.1, the graph vertical axis represents ground displacement (after instrument response removal), whereas, the graph horizontal axis represents the *sample index*, i.e., the position of each recorded data point in time. In the case of Figure 1.1.1, the sample index ranges from 0 to 6000, indicating that 6000 individual measurements were taken during the recording. The measurements are spaced according to the *sampling rate*, which represents how many data points are recorded per second. For instance, a sample rate of 100 Hz suggests that 100 samples are collected each second, so 6000 samples would correspond to 60 seconds of data. During a typical seismogram recording, the initial portion of the waveform contains only ambient seismic noise, here, up to approximately the first 400 samples. At this point, the primary (P) wave arrives, marked by a distinct burst of oscillations in the trace caused by ground reverberations. Around 1800 samples, a higher-amplitude secondary signal appears, corresponding to the shear (S) wave. Following this arrival, the signals gradually decay back toward the noise floor.

From each channel's 6000 sample trace, summary statistics such as the *peak amplitude*, *Root Mean Square (RMS) amplitude*, *spectral centroid* and *Signal-to-Noise Ratio (SNR)* can be obtained and can also be used in conjunction with the station coordinates and raw waveform metadata to predict the earthquake source. The peak amplitude is the single largest absolute excursion from zero i.e., the highest spike in the Z, N or E channel. The RMS amplitude is defined as the square root of the mean of squared amplitudes and it quantifies the traces' overall energy. The spectral cen-

troid, defined as the "centre of mass" of the spectrum, shows whether the energy is concentrated at lower or higher frequencies. Lastly, the SNR measures how clearly the seismic event stands out from ambient noise. The SNR is computed as:

$$SNR = \frac{RMS\ Amplitude_{signal}}{RMS\ Amplitude_{noise}}. \tag{1.1}$$

A numerical example of how these four features are computed can be found in Appendix A.

Having briefly introduced the topic of earthquake source characterization, the remainder of this chapter is divided in the following way. Section 1.2 delves into the different approaches that have been developed over the years to characterize seismic sources. Subsequently, the motivation and objective of study, namely, to characterize the sources of seismic events in the central Mediterranean region using a GNN framework, are discussed in Section 1.3. Following this, the development of GNNs is outlined in Section 1.4. Lastly, the dissertation structure is provided in Section 1.5.

1.2 Literature Review

Several approaches to earthquake source characterization have been introduced in the seismological literature, with the most well-established methods being *travel time-based inversion* (Z. Li & Van der Baan, 2016; Lin et al., 2015; H. Zhang & Thurber, 2003) and *waveform-based inversion* (Beskardes et al., 2017; Gajewski et al., 2007; Pesicek et al., 2014; Zhebel & Eisner, 2015). Travel time-based inversion methods employ a multi-step procedure to characterize earthquake sources. First, the arrival times of P-waves and S-waves are detected using phase detection techniques. Subsequently, these arrival times are attributed to specific earthquake events. Next, earthquake locations are determined by performing an inversion using the arrival times, station locations, and a velocity model. The earthquake magnitudes are then estimated based on waveform amplitudes and source-receiver distances. Although travel time-based inversion methods are commonly used in seismic applications, they are known to be susceptible to noise-related errors, specifically when estimating low-magnitude events, and may not fully utilize the phase and amplitude information within the complete waveform (X. Zhang et al., 2022). In contrast, waveform-based inversion methods combine full phase and amplitude information recorded by seismographs, leading to a higher-quality characterization of earthquake sources. Nevertheless, waveform-based inversion is computationally expensive. It is also important to mention that both

approaches require specialized knowledge in seismology.

A data-driven alternative for earthquake source characterization leverages waveform features using statistical learning techniques to enable the prediction of earthquake locations and magnitudes. Through advancements in technology, as well as the availability of high quality datasets, the implementation of statistical learning techniques has led to resounding success in seismological applications, such as *phase picking* (Zhu & Beroza, 2018), *seismic discrimination* (Z. Li et al., 2018), *waveform denoising* (Zhu et al., 2019), *earthquake location* (Perol et al., 2018), and *magnitude estimation* (Mousavi & Beroza, 2020b). Even though statistical learning methods have been applied to seismic event source characterization since the 1990s (Bladford, 1993), the first work to make use of more recent advances in deep learning was presented by Perol et al. (2018). In this work, a CNN was trained to detect earthquakes from the recordings of a single station to predict the source locations from within six regions. Despite the fact that this CNN model was successful in establishing foundational research in earthquake location detection via statistical learning methods, the approach was restricted to waveforms from a single seismic station and is only able to classify earthquakes into broad geographic groups, failing to provide specific location information. Since then, more advanced single-station methods that focused on improving location accuracy have been introduced. For example, a Bayesian NN was developed by Mousavi & Beroza (2020a) to learn epicentre distance, P-wave travel time and associated uncertainty using single-station data. This was a successful attempt, nevertheless, concurrently, various developments in multi-station statistical learning methods were also being made. For example, Kriegerowski et al. (2018) proposed a CNN architecture that merges information from three component waveforms from multiple stations to predict hypocentre location, producing more accurate source parameter estimates than single station methods. Moreover, X. Zhang et al. (2020) introduced an end-to-end Fully Convolutional Network (FCN) to predict the probability distribution of earthquake location directly from input data obtained from multiple stations. This was later extended by X. Zhang et al. (2021) to determine earthquake locations and magnitudes from continuous waveforms to be used for earthquake early warning. Additionally, a CNN framework was also adopted by Shen & Shen (2021) to extract location, magnitude and origin time from continuous waveforms recorded across a seismic network.

Despite multi-station approaches being an improvement on single station approaches, the utilization of standard convolutional layers limits the methods in various ways. For starters, CNNs are designed to operate on evenly spaced grids like images,

where information is exclusively shared between neighbouring grid positions. Secondly, CNNs require the station locations in the input data to be static to learn positional mapping. These assumptions pose issues as seismic networks are not regularly spaced and may even record information related to non-adjacent stations. Moreover, dynamic station input is highly desirable for earthquake source characterization due to station outages, the addition/removal of stations to seismic networks, and the ability to select a localized array for the detection of small-magnitude events.

To address this issue, many researchers have developed graph-based statistical learning methods. For instance, Münchmeyer et al. (2020) proposed an attention-based transformer neural network (an attention-based architecture) for earthquake early warning. Building on this, Münchmeyer et al. (2021), managed to predict hypocentres and magnitudes for earthquakes. While their approach proved effective, as well as successful in employing a multi-station approach that allows for dynamic inputs, it is widely acknowledged in the literature to be computationally intensive. In fact, the authors themselves limited their experiments to just 25 stations due to these high computational demands, although they do not provide a detailed complexity analysis. Van den Ende & Ampuero (2020) developed a multi-station earthquake source characterization method based on graph convolution in order to avoid high complexity for large seismic networks. In their method, Van den Ende & Ampuero (2020) model each station as a node in an edgeless graph, attaching to each node both its processed waveform feature vector and its geographic coordinates. Van den Ende & Ampuero (2020) then apply single-station convolution and aggregate via global pooling. In order to improve upon Van den Ende & Ampuero (2020)'s work Yano et al. (2021) developed a multi-station method in which edges are selected and held fixed for all inputs. Unfortunately, this resulted in the same limitation inherent to CNNs. McBrearty & Beroza (2022) introduced a GNN framework that makes use of multiple predefined graphs built on both labels and station locations. Because not every seismic event is recorded by all stations, McBrearty & Beroza (2022)'s network supports changes in the set of input stations, i.e., it can handle missing station data at inference time. However, it still only uses a limited subset of each station's waveform information. X. Zhang et al. (2022) proposed a Spatiotemporal Graph Neural Network (STGNN) that creates edges automatically by fusing waveform features and spatial information, namely, station locations. Their method aimed to utilize the full functionality of GNN while allowing for flexibility in the location and number of seismic stations.

1.3 Motivation and Objective of the Study

Earthquakes can cause man-made and natural structures, as well as the contents within them, to fail and fall, resulting in the injury or death of many people. The most devastating earthquakes in documented history were the 1556 Shaanxi earthquake (Chan, 1982), the 1976 Tangshan earthquake (Gere & Shah, 1980), the 1960 Chilean earthquake (National Oceanic and Atmospheric Administration, 1960) and the 2004 Indian Ocean earthquake (Lay et al., 2005), which led to the destruction of several buildings and structures, leaving civilians homeless and also indirectly leading to the death of hundreds of thousands of people.

As evident from these examples, most earthquakes occur within the Circum-Pacific Belt, more commonly known as the Ring of Fire. Nevertheless, earthquakes occur world-wide, with the Mediterranean region considered to be the second most active seismic zone (Institut de Ciències del Mar (CSIC), 2021). In fact, as will be seen in Chapter 5, throughout the years, several earthquakes have been recorded within the central Mediterranean region. Characterization of an earthquake source plays a crucial role in many seismic studies, such as, earthquake early warning, hazard assessment and emergency response (L. Li et al., 2020). For example, early warning systems aim to detect earthquakes immediately after rupture begins to warn regions that have not yet experienced the strong ground shaking. In this context, the rapid prediction of the epicentral location, depth and magnitude is crucial, as these parameters determine which areas are at risk, how intense the shaking is likely to be, and how much warning time is available before damaging seismic waves arrive. Furthermore, in the case of high earthquake magnitudes over 6 on the Richter scale, an earthquake close to the coast can trigger a tsunami. Key parameters for a tsunami, such as maximum Sea Surface wave Height (SSH), are strongly dependent on the earthquake source parameters. Consequently, the aim of this dissertation is to establish a statistical model, by combining different neural networks, in order to characterize the source of the earthquakes occurring in the central Mediterranean region, particularly focusing on the Maltese islands, Sicily and the surrounding areas (see Figure 1.3.1).

The models will be trained and validated on earthquake data recorded during the period 2013-2024, obtained from seismic stations positioned around the Maltese islands and Sicily, installed and maintained by the Istituto Nazionale di Geofisica e Vulcanologia (INGV) in Rome, Italy and the Seismic Monitoring and Research Group (SMRG) at the University of Malta. The target variables being considered in this study

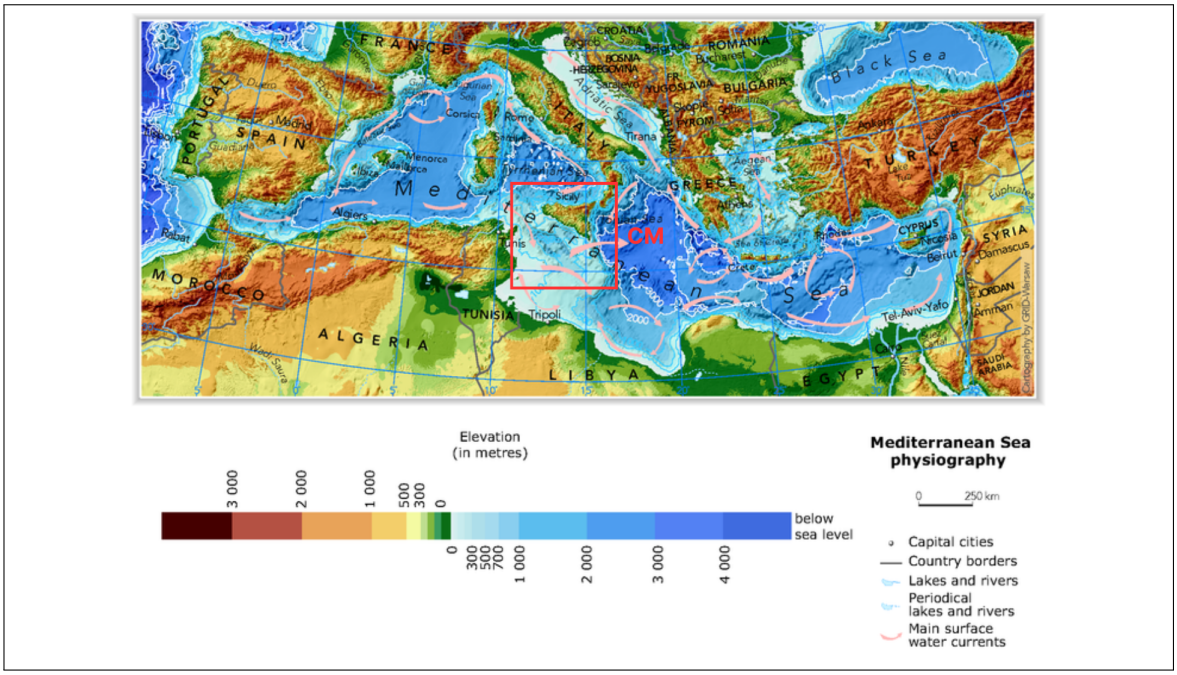

Figure 1.3.1: Bathymetric map of the study area, where CM denotes the Central Mediterranean (adapted from European Environment Agency (2009)).

are the latitude, longitude, depth and magnitude of the earthquake. These target variables will be predicted using station coordinates, raw waveform metadata, peak amplitude, root mean square amplitude, spectral centroid and SNR. More information regarding the data will be presented in Section 5.3.

The neural network architectures employed in this dissertation consist of a Multi-Layer Perceptron (MLP), a Convolutional Neural Network (CNN), and a Graph Neural Network (GNN). The MLP and CNN are primarily used as feature processing and representation-learning components, transforming station-level waveform and metadata into compact feature vectors. Each earthquake event is then represented as a graph, where vertices correspond to seismic stations and are associated with the learned feature representations. The GNN operates on this graph structure to integrate information across stations and produces the final outputs, namely the estimated latitude, longitude, depth, and magnitude of the earthquake source. It was decided to adopt a GNN framework as, according to the literature presented in Section 1.2, these NNs produced some of the most successful results. Two model architectures will be considered in the application, which mainly differ with regards to the input. Detail on the model architectures will be provided in Sections 5.5 and 5.6. The two architectures are compared using the performance metrics defined in Section 2.7, evaluated on the validation set, in order to identify the model configuration, and ultimately the architecture, that most effectively characterizes earthquake source parameters in the Central

Mediterranean region. Lastly, in Section 5.8.4, the resulting best model configurations are fitted on a test dataset spanning January 2025 to August 26, 2025, in order to examine predictive ability.

1.4 The Development of Graph Neural Networks

The development of GNNs is the result of the evolution of various types of neural networks. This section therefore provides a brief history of neural network architectures, beginning with the basic Artificial Neural Network (ANN), progressing through CNNs, and culminating in the emergence of GNNs.

1.4.1 Artificial Neural Networks

The conceptual roots of Neural Networks (NNs) can be traced back to the mid 20^{th} century, when researchers attempted to emulate how biological neurons in the human brain transmit and process information through the invention of simplified mathematical representations known as *artificial neurons*. In 1943, McCulloch & Pitts demonstrated how artificial neurons could be used to model simple logical functions. Nevertheless, the origins of NNs are often attributed to Rosenblatt (1958), who proposed assigning weights to the inputs of an artificial neuron and applying a threshold function to produce a single binary output. This binary classifier, known as the *perceptron*, laid the foundation for supervised learning in modern neural networks.

Despite being a groundbreaking development in computational models, the perceptron could not solve non-linear problems like image or speech recognition, and even struggled with basic tasks such as the logical '*exclusive or*' (Minsky & Papert, 1969). To address this issue, researchers began interconnecting neurons into layered architectures, forming the early versions of what are now known as ANNs. While this increased the representational capability of these neural networks, it was not until the development of the *backpropagation algorithm* by Rumelhart et al. (1986) that training such networks became possible. Building on this, Cybenko (1989) and Hornik (1991) later showed that given a sufficient number of neurons and appropriate activation functions, these neural networks can approximate any continuous function. These advancements marked the beginning of *deep learning*, where neural networks with multiple layers began to outperform traditional machine learning models on a variety of tasks.

As a result, ANNs became widely used for analysing Euclidean-structured data, such as fixed-length vectors in classification problems and tabular data in regression.

Nonetheless, since ANNs treat all input data vectors as independent and identically structured, they cannot account for spatial and structural properties of more complex data formats, such as images. Consequently, this motivated the evolution of more specialized neural network architectures such as those aimed at handling spatially and time dependent data.

1.4.2 Convolutional Neural Networks

CNNs extend ANNs by operating on data with a *grid-like topology*, such as two-dimensional images or one-dimensional audio-signals. This is achieved through two main innovations, namely, *local receptive fields* and *parameter sharing*. Local receptive fields allow the CNN to capture spatially local patterns. Parameter sharing ensures that the same pattern detector is used repeatedly across different parts of the input, significantly decreasing the amount of trainable parameters, as well as, enhancing the model's ability to generalize over similar patterns in varying positions.

CNNs are thought to have emerged from the *neocognitron*, which was proposed by Fukushima in 1980. Inspired by the structure of the brain's visual cortex, the neocognitron is a hierarchical multilayered NN capable of robust visual pattern recognition due to the network's invariance to shift in the position of the input patterns. Nevertheless, it was the *LeNet-5*, proposed by LeCun et al. (1998), that became the first notable CNN model.

Over the years, various researchers extended the basic CNN by proposing more advanced architectures designed to handle complex datasets and tasks. In fact, the first major break through for CNNs was the invention of the *AlexNet* architecture (Krizhevsky et al., 2012), which placed first in the ImageNet LSVRC-2010 contest, playing a crucial role in driving the utilization of deep learning in the industry. Besides this, AlexNet was one of the first NNs to take advantage of GPU acceleration for training deep networks, as well as introduced popular activation functions such as the Rectified Linear Unit (ReLU) function and various CNN regularization techniques. AlexNet then influenced the development of more advanced and modern architectures such as, *VGGNet* (Simonyan & Zisserman, 2015), *ResNet* (He et al., 2016), *GoogLeNet* (Szegedy et al., 2015) and *Xception* (Chollet, 2017), among others. Despite their success, CNNs are restricted to data that lie on regular grids and are not able to generalize to non-Euclidean domains such as, *molecular structures, transportation systems* or *seismic station networks*.

1.4.3 Graph Neural Networks

Real world systems, like molecular structures, transportation systems and seismic station networks, are best represented as *graphs*, where *entities* (nodes) and their *association* with each other (edges) vary in number and order. The traditional NNs discussed in Sections 1.4.1 and 1.4.2 are not designed to handle this data structure as they rely on fixed-dimensional inputs and grid-like data assumptions. This led to the need to extend NNs' functionality to cater for graph-structured data.

Early attempts applied *recursive neural networks* to directed acyclic graphs (Frasconi et al., 1998; Sperduti & Starita, 1997), facilitating information propagation through recursive formulations. Gori et al. (2005) then generalized recursive NNs to arbitrary graphs (i.e., not limited to directed acyclic graphs), introducing the first formal GNN framework. This development introduced an iterative fixed-point mechanism in which node representations were updated through what is known as repeated message passing until convergence. More information on message passing will be given in Section 4.3.1.

Despite this formulation significantly broadening the applicability of NNs to graph-structured data, it brought about new theoretical and computational challenges, specifically pertaining to the lack of guarantee of convergence and the possible instability of training. To tackle these problems, Scarselli et al. (2008) refined this framework, in turn improving the training procedure. Micheli (2009) then introduced a feedforward architecture, known as Neural Network for Graphs (NN4G), built to avoid the convergence limitations of fixed-point methods.

Around the same time, it was observed that the operations of CNNs could be interpreted through a graph-based perspective. For example, a 2D image could be considered as a grid graph where each pixel is a node linked to its neighbours (i.e., adjacent pixels). This realization motivated attempts at generalizing the notion of convolution to arbitrary graphs.

Besides the need to apply NNs to graph data, GNNs were also shaped by advancements in *graph representation learning* which is the process of learning continuos vector representations of nodes, edges and subgraphs. Up until then, conventional statistical learning methods used for graph analysis relied heavily on manually designed features, which were not only time consuming, but also domain specific. The success of representation learning in Natural Language Processing (NLP), specifically as a result of word embeddings, prompted similar approaches for graphs. One of the first algorithms to apply this idea was DeepWalk (Perozzi et al., 2014) by using random walks on graphs

to create sequences of nodes, which were then embedded utilizing methods borrowed from language modelling. This breakthrough gave rise to a wide range of methods such as node2vec (Grover & Leskovec, 2016), LINE (Tang et al., 2015) and TADW (Yang et al., 2015), improving the efficiency and quality of graph representations.

Unfortunately, these shallow representation methods suffered from two considerable limitations, namely, *scalability* and *generalization*. Nevertheless, GNNs overcame these drawbacks by combining the structural learning capability of CNNs with the flexibility of graph embeddings, forming trainable architectures that learn representations through aggregating and transforming information from local neighbourhoods.

One particular variant of GNNs is the EdgeConv network, first introduced by Wang et al. (2019). EdgeConv departs from fixed graph structures by rebuilding the connections between nodes at each layer based on their current feature representations. In practice, this means that every node "looks" for its nearest neighbours in the feature space and then learns richer edge features by comparing itself to those neighbours. These learned comparisons are then aggregated, typically by a simple maximum or average operation, to update each node's representation.

1.5 Outline of Dissertation

This chapter provided an introduction to earthquakes and the challenge of estimating their sources, along with a comprehensive literature review of the methods developed to address this problem. It also traced the evolution of ANNs, CNNs and GNNs. The following initial chapters will outline the theoretical underpinnings of these NN architectures. Subsequently, these theoretical concepts will be applied to characterize the seismic sources of the earthquakes occurring in the central Mediterranean region.

More specifically, Chapter 2 introduces fundamental concepts of ANNs, starting with the earliest and simplest network, the feedforward NN. Core ideas such as layers, neurons, activation functions, loss functions and trainable parameters are presented. The chapter then explains how a feedforward NN processes information during forward propagation, and how the backpropagation algorithm is then used to adjust the parameters by minimizing loss. The Universal Approximation Theorem (UAT), an important theoretical result which demonstrates that feedforward NNs are capable of approximating any continuous function, is also highlighted. Finally, the chapter provides a brief overview of regularization techniques conventionally used to improve ANN performance, followed by a discussion on the metrics that will be used to evaluate said

model performance.

Subsequently, Chapter 3 builds on the theory presented in Chapter 2 by focusing on CNNs, which play a crucial role in the application as they are used to process the raw waveform information. In this chapter, the specialized architecture components that distinguish CNNs from feedforward NNs, namely, convolutional layers, pooling layers and fully connected layers, and their contribution to enabling the CNN to effectively process grid-structured data is highlighted. The universality of CNNs is also discussed by extending the UAT to this architecture. Lastly, this chapter presents advanced regularization methods that adapt the conventional ANN regularization techniques to the CNN framework.

Following this, Chapter 4 develops the theoretical underpinnings of the GNN framework, starting with an introduction to fundamental graph concepts including the definition of graphs in terms of vertices and edges. This chapter is particularly important, as the dataset used in this work will be represented as a graph in which seismic stations correspond to vertices and edges define the relationships between them. An explanation on how information is shared between stations through the neural message passing paradigm, enabling the GNN to capture dependencies across graphs is given. Additionally, the approach taken to dynamically generate edges between vertices using a similarity criteria and k-Nearest Neighbours (KNNs), allowing the graph structure to adapt to the data, is also described. The chapter is concluded by discussing some universality results on graphs and set functions.

The concepts explained in Chapters 2-4 are then implemented in the application portion of this dissertation which is found in Chapter 5. Here the main contribution of this dissertation, namely, the prediction of the latitude, longitude, depth and magnitude, for events in the central Mediterranean region, is presented. Each target variable will be modelled as a graph, where seismic stations act as vertices and relationships between them are defined by edges. The chapter begins by detailing the computational setup of the analysis, including the hardware specifications of the local machine used, and the modules included in the Python environment. The central Mediterranean earthquake dataset is then introduced, together with the preprocessing steps applied prior to analysis.

Chapter 5 then proceeds to outline the model standards, training procedures, and evaluation criteria used throughout the analysis. Two-graph based model architectures are considered, specifically, an edgeless graph model and a dynamic edges GNN model,

with their architectures described in detail. Experiments for each model are conducted with the aim of improving final predictive performance. The results are presented and compared to identify which model architecture is best suited to the data. Following this, the most successful models are applied to unseen events to asses its ability to predict earthquake source 'parameters'. Finally, Chapter 6 summarizes the dissertation's key findings, discusses the challenges encountered, and offers recommendations for future research.

Chapter 2

Artificial Neural Networks

2.1 Introduction

Artificial Neural Networks (ANNs) are a large class of learning models that aim to cater for complex relationships in the data. Multi-Layer Perceptrons (MLPs) are the earliest form of these models. As datasets grew larger and tasks became more complex, various specialized extensions emerged. For instance, CNNs were invented to handle input data with a grid-like structure and GNNs were developed to cater for graph-structured data. Nevertheless, the core architecture of ANNs remains central to many modern neural network architectures. As a result, this chapter lays out the theoretical underpinnings of ANNs that will serve as the foundation for more advanced architectures.

Section 2.2.1 begins by introducing the concept of an *artificial neuron* which is the fundamental building block of the basic ANN, and by explaining how input data is fed forward through every layer of the network, in a procedure called *forward propagation*. Additionally, the Universal Approximation Theorem (UAT), a theoretical result which demonstrates that NNs are capable of approximating any continuous function, is presented. Sections 2.3 and 2.4 then explore important concepts such as the *learning rate* and *loss functions*. The loss function provides a measure of how far the network's predictions deviate from the expected output, while the learning rate determines the extent to which the network adjusts its parameter in response. Section 2.5 then delves into how the estimates of network parameters are adjusted through the *backpropagation algorithm*. Regularization techniques used to improve training are then discussed in Section 2.6. Finally, Section 2.7 presents the metrics that are used to evaluate a network's model performance.

2.2 An Overview of Artificial Neural Networks

Inspired by the human brain, ANNs are constructed from several interconnected nodes referred to as *(artificial) neurons* in three "blocks" of layers, namely the *input layer*, *hidden layer(s)* and *output layer*. Data enters the neural network through the input layer, it is then passed onto the hidden layers where it is processed via mathematical transformations. These hidden layers learn to extract patterns from the data, whether it is in its raw form or previously transformed information. After processing, the data is fed into the output layer, which generates the final predictions or classifications. In classification tasks, the number of output neurons corresponds to the number of classes, whereas prediction tasks typically use a single output neuron representing the variable of interest's prediction. As discussed in Chapter 1, the application in Chapter 5 deals with a prediction problem.

A NN with two or more hidden layers is called a "deep" neural network (Shrivastava, 2022). The neural networks that will be applied in the application in Chapter 5 are deep neural networks. Each hidden layer can capture different levels of abstraction, for instance in image data, lower layers detect simple patterns, such as *edges* and higher layers detect more complex ones, like *shapes*. Among the various architectures of neural networks, one of the most fundamental and widely studied is the feedforward neural network, a NN in which information passes strictly from input to output. This section will now delve deeper into the structure and processes of feedforward neural networks, and introduce the UAT, an important theorem within the NN framework.

2.2.1 Artificial Neurons and the Basic Neural Network

Artificial neurons transform a vector of real valued inputs into a single real-valued output. This output can in turn serve as input to other artificial neurons. The earliest artificial neurons could only handle binary inputs (McCulloch & Pitts, 1943), however these were then extended by weighting the inputs (Rosenblatt, 1958), forming what is known as a *perceptron* (see Figure 2.2.1).

In Figure 2.2.1, $\boldsymbol{x} = (x_1, ..., x_n)'$ is an n-vector of real-valued inputs with corresponding n-vector of weights $\boldsymbol{w} = (w_1, ..., w_n)'$. Furthermore, w_0 is referred to as the *bias* which is used to adjust the output along with the weighted sum of inputs and $\sigma(.)$ is an *activation function* that maps $\mathbb{R}$ onto some bounded interval, typically, $(-1, 1), (0, 1)$ or $[0, \infty)$ in order to generate an output $o(x_1, ..., x_n)$. Then, the output

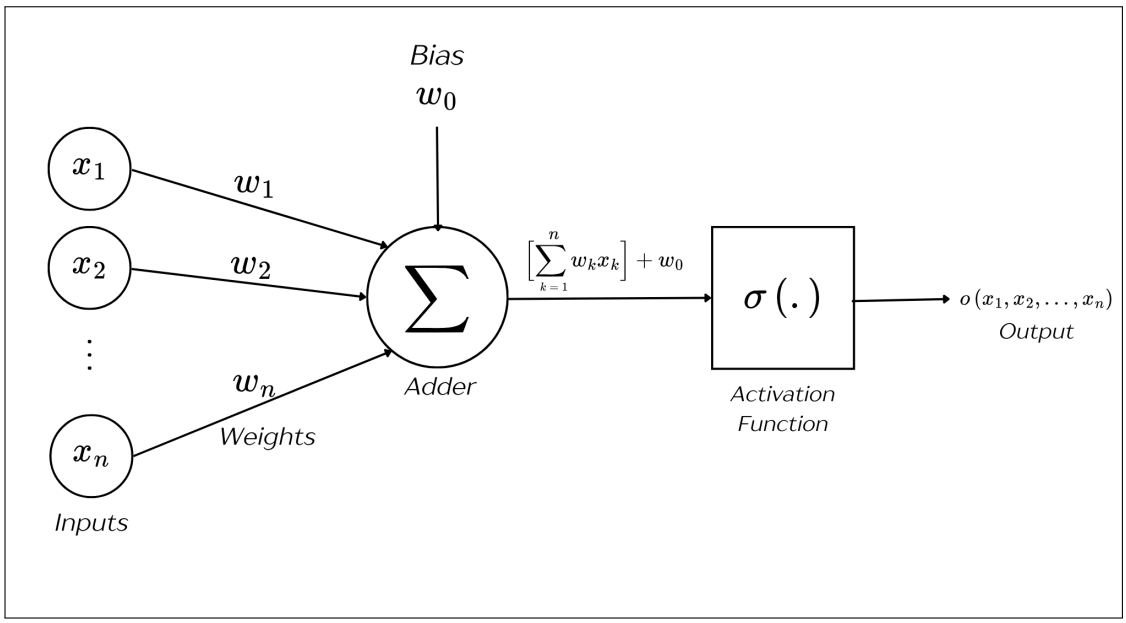

Figure 2.2.1: Structure of a *perceptron*.

$y = o(x_1, ..., x_n)$ produced by the perceptron is:

$$y = \sigma\left(\left[\sum_{k=1}^{n} w_k x_k\right] + w_0\right) = \sigma(z), \tag{2.1}$$

where z is the output of the neuron before the activation function is applied. There are several activation functions within the literature, the most common being the *logistic sigmoid function*, the *bi-polar sigmoid function* and the *hyperbolic tangent function*, which are defined in Equations (2.2)-(2.4), respectively.

$$\sigma_{logi}(z) = \frac{1}{1 + \exp(-z)}, \tag{2.2}$$

$$\sigma_{sigm}(z) = \frac{1 - \exp(-z)}{1 + \exp(-z)}, \tag{2.3}$$

$$\sigma_{tanh}(z) = \tanh(z). \tag{2.4}$$

The sigmoid functions and the hyperbolic tangent functions are monotonic and differentiable, which are desirable properties within the context of NNs due to the back-propagation algorithm, as will be seen in Section 2.5. However, activation functions, such as the sigmoid and hyperbolic tangent, can lead to the vanishing and exploding gradient problem. As will be seen in Section 2.5, during backpropagation, gradients are propagated through the network by repeatedly applying the chain rule, which involves

multiplying by the derivatives of the activation functions at each layer. For sigmoid and tanh, these derivatives become very small when the activations saturate at large positive or negative inputs, causing gradients to shrink as they propagate backwards through the network. As a result, the weights of earlier layers receive very small updates (vanishing gradients), while in some cases gradients may instead grow uncontrollably (exploding gradients), leading to unstable training.

These issues become more pronounced in deep networks, where many successive multiplications are performed during backpropagation. One way to mitigate this problem is through the use of the ReLU activation function (Tóth, 2013), defined in Equation (2.5), which does not saturate for positive inputs and therefore helps preserve gradient flow. In addition, vanishing and exploding gradients can be alleviated through appropriate weight initialization strategies (see Section 3.5.1).

$$\sigma_{ReLU}(z) = max\{z, 0\} = \begin{cases} z \text{ if } & z \geq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

The tanh and ReLU activation functions will be used in the application presented in Chapter 5. The sigmoid function is presented here as it was among the first activation functions proposed for artificial neurons and plays a central role in much of the theoretical framework discussed in Chapter 2.

One drawback of the *perceptron* in Figure 2.2.1 is its inability to solve non-linear problems such as image or speech recognition, or even basic tasks like the logical exclusive or (Minsky & Papert, 1969). To address this, researchers began interconnecting neurons into layered architectures, forming MLPs. Both the perceptron and the MLP fall under the category of feedforward NNs. A feedforward neural network is a type of ANN in which information flows in a single, forward direction, from the input layer, through any hidden layers, to the output, without any loops or feedback connections. The crucial difference between the perceptron and the MLP is that the perceptron consists of a single layer and is limited to linearly separable problems, whereas the MLP introduces hidden layers that enable the modelling of non-linear relationships. Up until now, simple notation has been used to explain the basic ideas of artificial neurons, weights, and activation functions. This notation will now be extended to better illustrate the more complex structure of an ANN, namely the *feedforward neural network*. Figure 2.2.2 shows an example of a feedforward neural network that contains an input layer with n_x neurons, a single hidden layer with n_h neurons, and an output

layer that generates a scalar output $\hat{y}$ for prediction problems. It is worth noting that the neural network in Figure 2.2.2 is a two-layer MLP, since the input layer is typically not included in the enumeration of the layers.

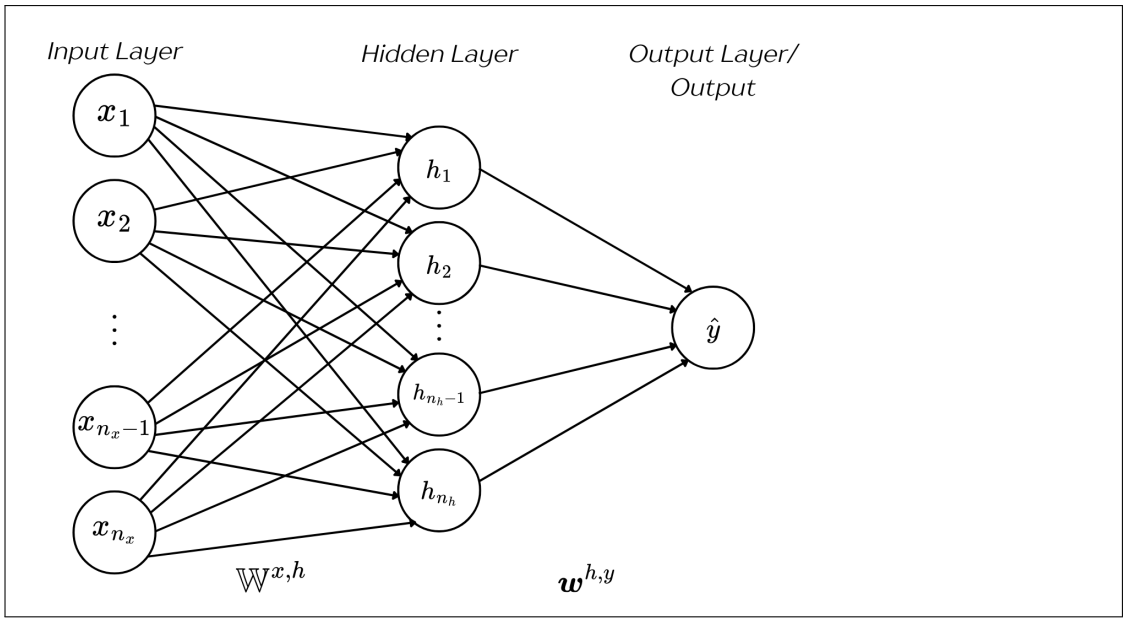

Figure 2.2.2: Example of a two-layer *feedforward* neural network (i.e., an MLP).

In Figure 2.2.2, the vector of inputs $\boldsymbol{x} = (x_1, ..., x_{n_x})'$ denotes a single row of the data matrix $\mathbb{X}$, containing the observed values for the sample being analysed. In practice, N vectors of observations will be available:

$$\mathbb{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n_x} \\ x_{21} & x_{22} & \dots & x_{2n_x} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & ... & x_{Nn_x} \end{bmatrix}.$$

Furthermore,

$$\mathbb{W}^{x,h} = \begin{bmatrix} w_{11}^{x,h} & w_{12}^{x,h} & \dots & w_{1n_h}^{x,h} \\ w_{21}^{x,h} & w_{22}^{x,h} & \dots & w_{2n_h}^{x,h} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_x1}^{x,h} & w_{n_x2}^{x,h} & ... & w_{n_xn_h}^{x,h} \end{bmatrix} \quad \text{and} \quad \boldsymbol{w}^{h,y} = \begin{bmatrix} w_1^{h,y} \\ w_2^{h,y} \\ \vdots \\ w_{n_h}^{h,y} \end{bmatrix}$$

are an $(n_x \times n_h)$-weight matrix and an n_h-weight vector respectively. These weight matrix and vector determine how strong the connection between the neurons of one layer and the neurons of the subsequent layer is. The $(i, i')^{\text{th}}$ element of $\mathbb{W}^{x,h}$ represents the weight joining the i^{th} input neuron to the i'^{th} hidden neuron, while, the i'^{th} element

of $\boldsymbol{w}^{h,y}$ represents the weight joining the $i^{'\text{th}}$ hidden neuron to the output $\hat{y}$. It is to be noted that a neuron in a layer might not necessarily be connected to each neuron in the succeeding layer. In such cases a zero weight is used in the connection between these two neurons. In addition to this, the bias terms corresponding to the layers are contained in the bias vector $\boldsymbol{w}_0^h = (w_0^{h_1}, ..., w_0^{h_{n_h}})'$ and in the scalar w_0^y.

For application purposes, the data matrix $\mathbb{X}$ is typically divided into two subsets: the *training set* and the *validation set*, with the validation set generally being smaller than the training set. A common approach is to randomly allocate approximately 80% of the observed data to the training set, leaving the remaining 20% for the validation set. As will be seen in Section 2.5, through the use of the training set, weights and biases are updated within the backpropagation algorithm. This is an iterative process referred to as *training*. The validation set is then used to tune any hyperparameters and monitor the model performance during training. A test set is then sometimes used to evaluate the final model's generalization performance and provide an unbiased estimate of how the model will perform on new, unseen data. With the structure of a basic feedforward neural network established, the forward propagation procedure will be explained next.

2.2.2 Forward Propagation

Feedforward neural networks undergo the process of *forward propagation*, whereby information is inserted into the input layer and passed onto the hidden layers through the use of weights, biases and activation functions, until the information reaches the output layer. This layer then generates the final output which for prediction type of problems takes the form of a scalar $\hat{y}$.

Consider a feedforward neural network consisting of an input layer, N_h hidden layers, and an output layer. Let $\boldsymbol{x}$ serve as input vector to the input layer, passing each of its components through its respective neurons. The input layer's output is then passed onto the first hidden layer, to produce the n_{h_1}-dimensional 'output':

$$\hat{\boldsymbol{o}}^{h_1} = \sigma^{h_1}\left((\mathbb{W}^{x,h_1})^\intercal \hat{\boldsymbol{o}}^x + \boldsymbol{w}_0^{h_1}\right) = \sigma^{h_1}(\boldsymbol{z}^{h_1}), \tag{2.6}$$

where σ^{h_1} is the activation function utilized in the first hidden layer h_1, $\mathbb{W}^{x,h_1}$ is an $(n_x \times n_{h_1})$-matrix containing the weights joining the input layer to the first hidden layer and $\boldsymbol{w}_0^{h_1}$ is an n_{h_1}-bias vector associated with the first hidden layer. It is worth mentioning that, typically, the components of the 'output' vectors $\boldsymbol{o}^{h_q}$, $q = 1, .., N_h$,

are referred to as *activations*. Subsequently, for each hidden layer h_q, $q = 2, ..., N_h$, an n_{h_q}-dimensional 'output' vector, $\hat{\boldsymbol{o}}^{h_q}$ can be obtained as follows:

$$\hat{\boldsymbol{o}}^{h_q} = \sigma^{h_q}\left((\mathbb{W}^{h_{q-1},h_q})^\intercal \hat{\boldsymbol{o}}^{h_{q-1}} + \boldsymbol{w}_0^{h_q}\right) = \sigma^{h_q}(\boldsymbol{z}^{h_q}), \tag{2.7}$$

where σ^{h_q} is the activation function used in hidden layer h_q, $\mathbb{W}^{h_{q-1},h_q}$ is an $(n_{h_{q-1}} \times n_{h_q})$-matrix containing the weights joining hidden layer h_{q-1} to hidden layer h_q, and $\boldsymbol{w}_0^{h_q}$ is a n_{h_1}-bias vector associated with hidden layer h_q. Then, by feeding the 'output' $\hat{\boldsymbol{o}}^{h_{N_h}}$ into the output layer, the final output $\hat{y}$ is obtained using:

$$\hat{y} = \sigma^y\left((\boldsymbol{w}^{h_{N_h},y})^\intercal \hat{\boldsymbol{o}}^{h_{N_h}} + w_0^y\right) = \sigma^y(z^y), \tag{2.8}$$

where σ^y is the activation function utilized in the output layer, $\boldsymbol{w}^{h_{N_h},y}$ is an n_{N_h}-vector containing the weights joining the final hidden layer to the output neuron, and w_0^y is the bias term. The final output $\hat{y}$ can then be any real-valued scalar that falls within the range of the corresponding activation function.

2.2.3 The Universal Approximation Theorem

The *representation problem* is a prominent topic discussed within the context of neural networks (Rumelhart et al., 1995). It was briefly introduced in Section 2.2.1 when it was seen that the main reason for the invention of NNs at the time was to be able to approximate any continuous function (Cybenko, 1989). The ability of neural networks to approximate functions is formally established through the Universal Approximation Theorem (UAT) which provides theoretical assurance that, under certain conditions, any continuous function defined on some compact subset of $\mathbb{R}^n$ can be approximated using a feedforward NN containing one hidden layer and a finite number of neurons. Before presenting the UAT, it is necessary to define a *discriminatory function* as it plays a crucial role in the formulation, as well as the proof of the theorem:

Definition 1 Discriminatory Function
Let $\boldsymbol{I}_{n_x}$ be an n_x-dimensional neuron cube, $[0,1]^{n_x}$ and $M(\boldsymbol{I}_{n_x})$ represent the space of finite, signed regular Borel measures on $\boldsymbol{I}_{n_x}$. A function σ is said to be discriminatory if for a measure $\mu \in M(\boldsymbol{I}_{n_x})$,

$$\int_{\boldsymbol{I}_{n_x}} \sigma(\boldsymbol{w}^T\boldsymbol{x} + w_0)d\mu(\boldsymbol{x}) = 0 \implies \mu = 0, \quad \forall \boldsymbol{w}, \boldsymbol{x} \in \mathbb{R}^{n_x}, w_0 \in \mathbb{R}$$

With the definition of a discriminatory function established, the UAT can be stated in Theorem 1. Any mathematical results from functional analysis required to prove the UAT can be found in Appendix B.

Theorem 1 Universal Approximation Theorem for Feedforward Neural Networks (Cybenko, 1989)

Let the activation function $\sigma(.)$ be chosen in such a way that it is a discriminatory function. Furthermore, suppose that $C(\boldsymbol{I}_{n_x})$ represents the space of continuous functions on $\boldsymbol{I}_{n_x}$. Letting $\boldsymbol{w}_i, \boldsymbol{x} \in \mathbb{R}^{n_x}, \varsigma_i, w_{i0} \in \mathbb{R}$ be constant, for $i = 1, \ldots, n_h$, where n_h denotes the number of neurons in the hidden layer, finite sums of the following form:

$$F(\boldsymbol{x}) = \sum_{i=1}^{n_h} \varsigma_i \sigma(\boldsymbol{w}_i^T \boldsymbol{x} + w_{0i}) \tag{2.9}$$

are said to be dense in $C(\boldsymbol{I}_{n_x})$ and for any $f \in C(\boldsymbol{I}_{n_x}), \varepsilon > 0, \exists F(\boldsymbol{x})$ s.t.

$$|F(\boldsymbol{x}) - f(\boldsymbol{x})| < \varepsilon \quad \forall \boldsymbol{x} \in \boldsymbol{I}_{n_x}.$$

Proof. Suppose $\sigma(\cdot)$ is a discriminatory function as defined in Definition 1. Let $S \subset C(\boldsymbol{I}_{n_x})$ be the set of all functions of the form $F(\boldsymbol{x})$ as in Equation (2.9). By definition, S is a linear subspace of $C(\boldsymbol{I}_{n_x})$. The aim is to prove that the closure of S is all of $C(\boldsymbol{I}_{n_x})$ (i.e., $\bar{S} = C(\boldsymbol{I}_{n_x})$).

Suppose that the closure of S is not all of $C(\boldsymbol{I}_{n_x})$. Instead, the closure of S, denoted $\bar{S}$, is a closed normed proper subspace of $C(\boldsymbol{I}_{n_x})$, where $C(\boldsymbol{I}_{n_x})$ is equipped with the sup norm. Then by the Hahn-Banach theorem (Banach, 1929a,b; Hahn, 1927), if $\bar{S}$ is a closed proper subspace of $C(\boldsymbol{I}_{n_x})$, $\exists$ a bounded linear functional $\mathcal{L}$, $\mathcal{L} : C(\boldsymbol{I}_{n_x}) \to \mathbb{R}$, such that

$$\mathcal{L}(\psi) \neq 0, \ \psi \in \bar{S} \quad \text{but} \quad \mathcal{L}(g) = 0, \ \forall g \in \bar{S}. \tag{2.10}$$

By the Riesz-Markov-Kakutani Representation Theorem (Kakutani, 1941; Markov, 1938; Riesz, 1909), this bounded linear functional, $\mathcal{L}$, is of the form:

$$\mathcal{L}(\psi) = \int_{\boldsymbol{I}_{n_x}} \psi(\boldsymbol{x}) d\mu(\boldsymbol{x}), \tag{2.11}$$

for some $\mu \in M(\boldsymbol{I}_{n_x})$, $\forall \psi \in C(\boldsymbol{I}_{n_x})$. Every function of the form $\boldsymbol{x} \mapsto \sigma(\boldsymbol{w}^T \boldsymbol{x} + w_0)$ lies

in $S \subset \bar{S}$, hence

$$\int_{\boldsymbol{I}_{n_x}} \sigma(\boldsymbol{w}^T \boldsymbol{x} + w_0) d\mu(\boldsymbol{x}) = 0, \quad \forall \boldsymbol{w} \in \mathbb{R}^{n_x}, w_0 \in \mathbb{R}. \tag{2.12}$$

Since σ is discriminatory, it implies that $\mu = 0$. Therefore, $\mathcal{L}(\psi) = \int \psi d\mu = 0$, $\forall \psi \in C(\boldsymbol{I}_{n_x})$, so, $\mathcal{L}$ is a zero functional, contradicting the condition in 2.10. Hence, the assumption was false, and $\bar{S} = C(\boldsymbol{I}_{n_x})$. Equivalently, S is dense in $C(\boldsymbol{I}_{n_x})$ i.e., $\forall f \in C(\boldsymbol{I}_{n_x})$ and $\varepsilon > 0$, $\exists$ an $F \in S$, such that $||F(\boldsymbol{x}) - f(\boldsymbol{x})|| < \varepsilon$. □

Early universality results showed that networks with sigmoidal activation functions are universal approximators (Cybenko, 1989), which was later generalised to any non-constant, bounded, and continuous activation function (Hornik, 1991). This was further refined by Leshno et al. (1993), who proved that a necessary and sufficient condition for MLPs to be universal approximators is that the activation function is non-polynomial. While the UAT guarantees that any continuous function can be approximated by a NN, this may in practice require a very large number of neurons (Mitchell, 1997).

When these results were established, neural networks were typically shallow, often with a single hidden layer, which could require thousands of neurons to approximate complex functions. In contrast, deep architectures can represent the same functions far more efficiently; for example, a function that might require 1,000 neurons in a single hidden layer can often be approximated using around 10 neurons per layer across five or six layers. Thus, although shallow networks are theoretically universal, deep networks offer a much more compact and practical representation.

It is also worth mentioning that, UAT presented in this section relates to feedforward NNs. Over the years, there have been various generalizations of the original UAT to accommodate the architectures of more advanced neural networks such as CNNs and GNNs. The UAT for CNNs is presented in Section 3.4. Theory of universality of GNNs is presented in Section 4.5.

2.3 The Learning Rate

The learning rate, α_t, is a positive value that can be modified and lies within the range (0,1), where t denotes the iteration number of the backpropagation algorithm (see Section 2.5). It is a crucial component of any NN as it determines the rate at which the model adapts to the given problem. An adequate learning rate ensures that

during the training phase, weights and biases are updated accordingly so that there is fast convergence, as well as stability. If the learning rate chosen is too low, then the loss value will converge slowly or get stuck in a local minima. Conversely, a value for α_t which is too high might result in divergence (Bengio, 2012). This can be seen in Figure 2.3.1. Note that for simplicity of explanation a convex loss function is being considered, whereas in practice, loss functions are usually non-convex.

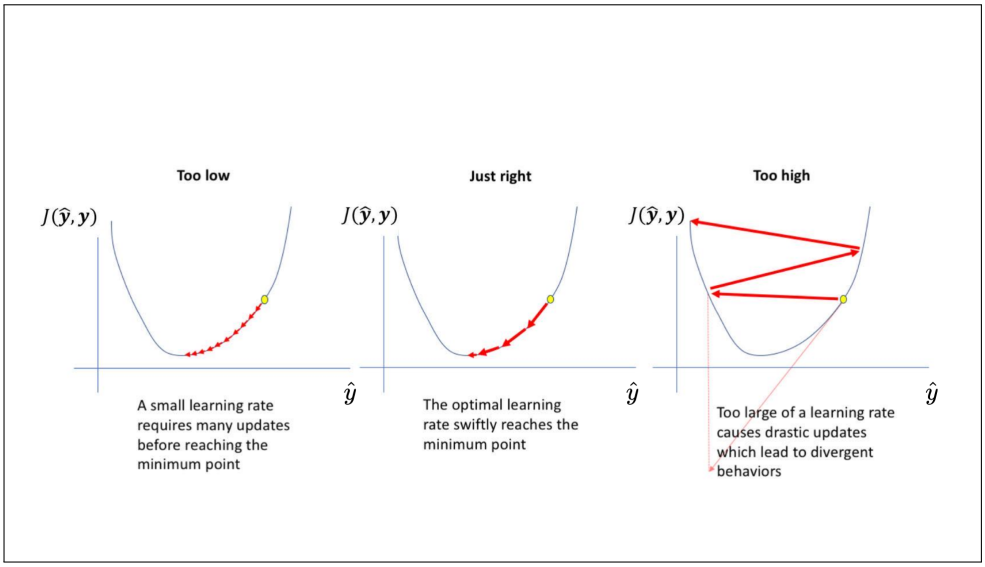

Figure 2.3.1: Different Learning Rates and their Effect on Convergence (adapted from Jordan, J. (2018)).

It is worth noting that different tasks and datasets may require different learning rates in order to achieve optimal performance. In the application presented in Chapter 5, the learning rate will be determined through hyperparameter tuning. During training, the learning rate can be systematically adjusted over time using *learning rate scheduling*. Rather than keeping the learning rate constant throughout training, a scheduler dynamically modifies it based on a predefined strategy or model performance. Learning rate schedulers aid training by accelerating learning in the early stages by using a higher learning rate and refining convergence in later stages by reducing the learning rate. Furthermore, learning rate schedulers help to mitigate certain issues like overfitting, oscillations or stagnation in training. Learning rate schedulers will not be elaborated on further, as they will not be implemented in the application.

2.4 Loss Functions

In the context of NNs, loss functions measure how well the NN performs by examining some function of the difference between the actual output $\boldsymbol{y} = (y_1, ..., y_{N_{train}})'$ and the predicted output $\hat{\boldsymbol{y}} = (\hat{y}_1, ..., \hat{y}_{N_{train}})'$ generated by the NN. Here N_{train} is the size of the

training set. When fitting a neural network model, the aim is to minimize the specified loss function which is in terms of the difference between the respective components of $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$.

There are different types of loss functions depending on which type of problem is being tackled. For prediction problems, the *Mean Absolute Error (MAE)* loss function $J_{MAE}(\hat{\boldsymbol{y}}, \boldsymbol{y})$, presented in Equation (2.13), the *Mean Squared Error (MSE)* loss function $J_{MSE}(\hat{\boldsymbol{y}}, \boldsymbol{y})$, presented in Equation (2.14), and the *Huber* loss function $J_H(\hat{\boldsymbol{y}}, \boldsymbol{y})$ (Huber, 1992) given by Equation (2.15) are typically used:

$$J_{MAE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{N_{train}} \sum_{k=1}^{N_{train}} |\hat{y}_k - y_k|, \tag{2.13}$$

$$J_{MSE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{N_{train}} \sum_{k=1}^{N_{train}} (\hat{y}_k - y_k)^2, \tag{2.14}$$

$$J_H(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{N_{train}} \sum_{k=1}^{N_{train}} \mathcal{J}_H(\hat{y}_k, y_k). \tag{2.15}$$

Here

$$\mathcal{J}_H(\hat{y}_k, y_k) = \begin{cases} \frac{1}{2}(\hat{y}_k - y_k)^2 & \text{if} \quad |\hat{y}_k - y_k| \leq \gamma \\ \gamma(|\hat{y}_k - y_k| - \frac{1}{2}\gamma) & \text{otherwise.} \end{cases}$$

The parameter γ is arbitrary in the literature related to statistical learning. As γ approaches infinity, the Huber loss function approximates the MAE loss function, while when γ tends to zero, it approximates the MSE loss function.

The specified loss function is minimized during the training phase using an iterative optimization technique referred to as the *gradient descent*. Consider the NN in Figure 2.2.2 along with a general loss function $J(\hat{\boldsymbol{y}}, \boldsymbol{y})$. The weights in the matrix $\mathbb{W}^{x,h}$ and the biases in the vector $\boldsymbol{w}_0^h$ at the t^{th} iteration of the gradient descent method are updated in the following way:

$$\mathbb{W}_{(t)}^{x,h} = \mathbb{W}_{(t-1)}^{x,h} - \alpha_t \frac{\partial J(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial \mathbb{W}_{(t-1)}^{x,h}}, \tag{2.16}$$

$$\boldsymbol{w}_{0(t)}^h = \boldsymbol{w}_{0(t-1)}^h - \alpha_t \frac{\partial J(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial \boldsymbol{w}_{0(t-1)}^h}, \tag{2.17}$$

where α_t is the *learning rate* which was discussed in Section 2.3. Moreover, $\mathbb{W}_{(t)}^{x,h}$ is

$\mathbb{W}^{x,h}$ updated at the t^{th} iteration and $\mathbb{W}^{x,h}_{(t-1)}$ denotes the weight matrix from the prior iteration. Note that $\boldsymbol{w}^{h}_{0(t)}$ and $\boldsymbol{w}^{h}_{0(t-1)}$ are defined in a similar manner. The weight vector $\boldsymbol{w}^{h,y}$ and the bias w^{y}_{0} may be updated in a similar manner.

Unfortunately, due to the calculation of the partial derivatives with respect to each parameter, the gradient descent method is computationally intensive. In order to address such an issue, the Stochastic Gradient Descent (SGD) method was invented as an approximation to the gradient descent method (Kiefer & Wolfowitz, 1952; Robbins & Monro, 1951). The SGD method estimates the gradient in a stochastic way through randomly choosing a small subset of the training data and calculating the gradients based on that subset. As a result, the SGD method completes iterations more quickly, however, it converges at a slower rate (Bottou, 2012).

In practice, the reasoning behind the gradient descent method and its variants is implemented through tools known as *optimizers*. Given the gradients, computed via backpropagation (see Section 2.5), and a learning rate, the optimizer applies the appropriate update rule to adjust the model's weights. Although SGD itself is an optimizer, various enhancements have been proposed to improve convergence speed and stability. These include SGD with momentum, RMSProp, Adam and AdamW among others.

SGD with momentum is similar to the traditional SGD method mentioned earlier, however it employs an exponentially weighted average of the gradients $\frac{\partial J(\hat{\boldsymbol{y}},\boldsymbol{y})}{\partial w^{x,h}_{ii'(t)}}$. Through this, the algorithm converges to the loss function minimum at a quicker rate than the conventional SGD (Kingma & Ba, 2017). This can also be seen in Figure 2.4.1. SGD with momentum has the following formulation:

$$w^{x,h}_{ii'(t+1)} = w^{x,h}_{ii'(t)} - \alpha_t \left(\rho m_{t-1} + (1+\rho) \left[\frac{\partial J(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial w^{x,h}_{ii'(t)}} \right] \right), \tag{2.18}$$

where $w^{x,h}_{ii'(t)}$ is the $(i,i')^{\text{th}}$-component of $\mathbb{W}^{x,h}_{(t)}$ and α_t is the learning rate. In addition to this, m_t denotes the momentum term at time t and ρ is a hyperparameter referred to as the momentum coefficient. Typically, m_t takes initial value 0 and ρ lies within the range $(0,1)$. Setting the momentum coefficient ρ to 0 results in the conventional SGD, whereas a value of 1 indicates no decay. Consequently, the value for ρ is usually set to some constant value between 0 and 1.

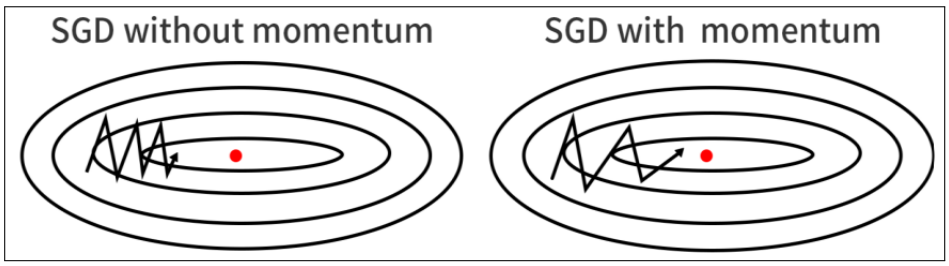

Figure 2.4.1: Convergence Rates for SGD and SGD with momentum (Du, 2019).

Another method that allows for fast convergence is RMSProp. Through this method, the learning rate is tweaked automatically using the following formula:

$$w^{x,h}_{ii'(t+1)} = w^{x,h}_{ii'(t)} - \frac{\alpha_t}{\sqrt{v_t + \varepsilon}} \frac{\partial J(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial w^{x,h}_{ii'(t)}}, \tag{2.19}$$

where $v_t = \beta v_{t-1} + (1-\beta) \left[\frac{\partial J(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial w^{x,h}_{ii'(t)}} \right]^2$ taking initial value 0. Here β is a hyperparameter commonly set to 0.9 and α_t represents the learning rate. Moreover, ε is a negligible constant, usually taken to be 10^{-8} (Goodfellow et al., 2016).

Combining the SGD with momentum method and RMSProp method results in the Adam optimizer. This method can be summarized by the following equations:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) \frac{\partial J(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial w^{x,h}_{ii'(t)}}, \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) \left[\frac{\partial J(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial w^{x,h}_{ii'(t)}} \right]^2, \end{aligned} \tag{2.20}$$

where $w^{x,h}_{ii'(t)}$ is as defined before. Furthermore, β_1 and β_2 are the decay rates for the moving averages of the *first* and *second* moments of the gradients, denoted by m_t and v_t, respectively. It is convention to set $\beta_1 = 0.9$ and $\beta_2 = 0.999$ (Kingma & Ba, 2017). Given that m_t and v_t are initially set to 0, and the hyperparameters β_1 and β_2 approach 1, this introduces a bias in the estimates of the first and second moments during the early steps of training. To counteract this, the Adam optimizer applies bias correction, yielding $\hat{m}_t$ and $\hat{v}_t$ via the following equations:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \quad \& \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}, \tag{2.21}$$

where β_1^t and β_2^t represent β_1 and β_2 to the power of t. The corrections $\hat{m}_t$ and $\hat{v}_t$ are updated automatically and for each parameter, the Adam Optimizer is evaluated in

the following way:

$$w^{x,h}_{ii'(t+1)} = w^{x,h}_{ii'(t)} - \hat{m}_t \left(\frac{\alpha_t}{\sqrt{\hat{v}_t - \varepsilon}} \right), \tag{2.22}$$

where $w^{x,h}_{ii'(t)}$, $\hat{m}_t$, $\hat{v}_t$, α_t and ε are as defined before.

Even though adaptive methods such as Adam have become a default method of choice for training NNs (Radford et al., 2016; Xu et al., 2015) due to their fast convergence, these methods do not generalize as well as SGD with momentum when applied to diverse deep learning tasks (Loshchilov & Hutter, 2019). Loshchilov & Hutter (2019) demonstrated that the main reason behind the poor generalization ability of the Adam optimizer is because the L_2 regularization (see Appendix C, Section C.1) is not nearly as effective for it as for SGD. The term *regularization* loosely refers to techniques used to prevent overfitting by discouraging overly complex models, and a formal definition will be provided in Section 2.6. In order to improve regularization in the Adam optimizer, Loshchilov & Hutter (2019) developed an extension known as the AdamW optimizer, in which *weight decay* is *decoupled from the gradient-based update* as follows:

$$w^{x,h}_{ii'(t+1)} = w^{x,h}_{ii'(t)} - \eta_t \left(\frac{\alpha_t \hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}} + \lambda w^{x,h}_{ii'(t)} \right), \tag{2.23}$$

where η_t is a scaling factor and $\lambda \in \mathbb{R}$ is a parameter representing weight decay. Furthermore, $w^{x,h}_{ii'(t)}$, $\hat{m}_t$, $\hat{v}_t$, α_t and ε are defined as before. In this dissertation, the AdamW optimizer will be employed as it generalizes better to unseen data than the conventional Adam optimizer.

It is worth noting that some optimizers, such as RMSProp and Adam, adapt the parameter learning rate internally. However, the internal adaptation is still dependent on a global learning rate hyperparameter, which plays a critical role in controlling the overall update magnitude.

The subsequent section will detail how the gradients, $\frac{\partial J(\hat{\boldsymbol{y}},\boldsymbol{y})}{\partial \mathbb{W}^{x,h}_{(t)}}$, $\frac{\partial J(\hat{\boldsymbol{y}},\boldsymbol{y})}{\partial \boldsymbol{w}^{h}_{0(t)}}$, $\frac{\partial J(\hat{\boldsymbol{y}},\boldsymbol{y})}{\partial \boldsymbol{w}^{h,y}_{(t)}}$ and $\frac{\partial J(\hat{\boldsymbol{y}},\boldsymbol{y})}{\partial w^{y}_{0(t)}}$, used in the gradient descent algorithm are calculated at each iteration t via the backpropagation algorithm.

2.5 The Backpropagation Algorithm

The *backpropagation algorithm* trains NNs by minimizing the loss between the predicted output and the actual output. Intuitively, the backpropagation algorithm can be seen as working in reverse order to the forward propagation procedure described in Section 2.2.2. The algorithm starts from the NN output and moves backward towards the input layer. At each step, it calculates the partial derivatives $\frac{\partial J(\hat{\boldsymbol{y}},\boldsymbol{y})}{\partial \mathbb{W}^{x,h}_{(t)}}$, $\frac{\partial J(\hat{\boldsymbol{y}},\boldsymbol{y})}{\partial \boldsymbol{w}^{h}_{0(t)}}$, $\frac{\partial J(\hat{\boldsymbol{y}},\boldsymbol{y})}{\partial \boldsymbol{w}^{h,y}_{(t)}}$ and $\frac{\partial J(\hat{\boldsymbol{y}},\boldsymbol{y})}{\partial w^{y}_{0(t)}}$, and using these derivatives the NN weights and biases are then updated through the gradient descent algorithm using Equations (2.16) and (2.17).

The derivative of the loss function with respect to the output layer's weights, $w^{h,y}_{ii'}$ can be computed using the calculus chain rule as seen in Equation (2.24):

$$\frac{\partial J(\hat{\boldsymbol{y}},\boldsymbol{y})}{\partial \boldsymbol{w}^{h,y}_{(t)}} = \frac{\partial J(\hat{\boldsymbol{y}},\boldsymbol{y})}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{y}} \frac{\partial z^{y}}{\boldsymbol{w}^{h,y}_{(t)}}. \tag{2.24}$$

For simplicity, in what follows, the subscript (t) will be dropped and so, without loss of generality, the backpropagation algorithm will be described at the t^{th} iteration. The first component of the Right Hand Side (RHS) of Equation (2.24), $\frac{\partial J(\hat{\boldsymbol{y}},\boldsymbol{y})}{\partial \hat{y}}$, can be obtained by differentiating the chosen loss function $J(\hat{\boldsymbol{y}},\boldsymbol{y})$ with respect to the predicted output $\hat{y}$. Recall from Section 2.2.2 that $\hat{y} = \sigma^{y}(z^{y})$ and so, given an activation function $\sigma(.)$, $\frac{\partial \hat{y}}{\partial z^{y}}$ can be worked out as follows:

$$\frac{\partial \hat{y}}{\partial z^{y}} = \frac{\partial \sigma^{y}(z^{y})}{\partial z^{y}}. \tag{2.25}$$

Lastly, the third term in the RHS of Equation (2.24) is given by:

$$\frac{\partial z^{y}}{\partial \boldsymbol{w}^{h,y}} = \sigma^{h}(\boldsymbol{z}^{h}) = \hat{\boldsymbol{o}}^{h}. \tag{2.26}$$

Thus, combining $\frac{\partial J(\hat{\boldsymbol{y}},\boldsymbol{y})}{\partial \hat{y}}$ with Equations (2.25) and (2.26) we obtain:

$$\frac{\partial J(\hat{\boldsymbol{y}},\boldsymbol{y})}{\boldsymbol{w}^{h,y}} = \frac{\partial J(\hat{\boldsymbol{y}},\boldsymbol{y})}{\partial \hat{y}} \frac{\partial \sigma^{y}(z^{y})}{\partial z^{y}} \hat{\boldsymbol{o}}^{h}. \tag{2.27}$$

In addition to this, we may express the derivative of the loss function with respect to the output layer's bias, w^{y}_{0} as a product of three derivatives using the chain rule:

$$\frac{\partial J(\hat{\boldsymbol{y}},\boldsymbol{y})}{\partial w^{y}_{0}} = \frac{\partial J(\hat{\boldsymbol{y}},\boldsymbol{y})}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{y}} \frac{\partial z^{y}}{w^{y}_{0}}. \tag{2.28}$$

The first two terms of the RHS of Equation (2.28) are exactly the same as in Equation (2.27). Moreover, the third term of the RHS of Equation (2.28) is simply a vector of ones. Thus,

$$\frac{\partial J(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial w_0^y} = \frac{\partial J(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial \hat{y}} \frac{\partial \sigma^y(z^y)}{\partial z^y}. \tag{2.29}$$

Having obtained $\frac{\partial J(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial \boldsymbol{w}^{h,y}}$ and $\frac{\partial J(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial w_0^y}$ at the t^{th} iteration, the gradient descent algorithm equations similar to Equations (2.16) and (2.17) are applied to the output layer weights and biases.

The backpropagation algorithm then proceeds by finding the partial derivatives of the loss function with respect to the hidden layer weights and biases. Like this, the weights and biases can be updated using the gradient descent algorithm. At the t^{th} iteration, the required partial derivatives are $\frac{\partial J(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial \mathbb{W}^{x,h}}$ and $\frac{\partial J(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial \boldsymbol{w}_0^h}$.

Once again, these partial derivatives may be expressed as a product of different terms using the chain rule as follows:

$$\begin{aligned}
\frac{\partial J(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial \mathbb{W}^{x,h}} &= \frac{\partial J(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial \hat{\boldsymbol{o}}^h} \frac{\partial \hat{\boldsymbol{o}}^h}{\partial \boldsymbol{z}^h} \frac{\partial \boldsymbol{z}^h}{\partial \mathbb{W}^{x,h}} \\
&= \frac{\partial J(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial \hat{\boldsymbol{o}}^h} \frac{\partial \hat{y}}{\partial z^y} \frac{\partial z^y}{\partial \hat{\boldsymbol{o}}^h} \frac{\hat{\boldsymbol{o}}^h}{\partial \boldsymbol{z}^h} \frac{\partial \boldsymbol{z}^h}{\partial \mathbb{W}^{x,h}} \\
&= \frac{\partial J(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial \hat{\boldsymbol{o}}^h} \boldsymbol{w}^{h,y} \frac{\partial \sigma^h(\boldsymbol{z}^h)}{\partial \boldsymbol{z}^h} \boldsymbol{x}
\end{aligned} \tag{2.30}$$

and

$$\begin{aligned}
\frac{\partial J(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial \boldsymbol{w}_0^h} &= \frac{\partial J(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial \hat{\boldsymbol{o}}^h} \frac{\partial \hat{\boldsymbol{o}}^h}{\partial \boldsymbol{w}_0^h} \\
&= \frac{\partial J(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial \hat{\boldsymbol{o}}^h} \boldsymbol{w}^{h,y} \frac{\partial \sigma^h(\boldsymbol{z}^h)}{\partial \boldsymbol{z}^h}.
\end{aligned} \tag{2.31}$$

The hidden layer weights and biases are once again updated through the gradient descent algorithm using Equations (2.16) and (2.17).

2.6 Regularization

Sometimes ANN models are not able to generalize well to new data as they suffer from over-fitting. In order to overcome this issue, regularization techniques may be employed during the training process. There are numerous types of regularization methods that

may be utilized within the NN framework. The methods relevant to the application in Chapter 5 are *early stopping* which will be covered shortly and *dropout regularization* which will be covered in Section 3.5.2. Theory regarding L_1 *and* L_2 *regularization* and *batch normalization* can be found in Appendix C.

Early stopping is a regularization method that monitors a model's performance during training at each epoch, using metrics such as validation loss or validation accuracy, and stops the training when the model's performance begins to worsen. In reality, the training is not terminated as soon as the validation performance stops improving. Instead it waits for a certain number of epochs. This is termed as *patience*.

The training process is stopped if the model performance does not get better within the patience window. For example, suppose the patience parameter is set to 10. This means that the training continues for another 10 epochs after the last best performance in order to check whether validation improves. If it does, the patience counter resets. On the other hand, if it does not improve, the training is stopped.

Early stopping reduces over-fitting by finding a balance between training the data adequately and maintaining the ability to generalize to new data. In this way, less parameters need to be updated, which ensures computational efficiency. A graphical representation of early stopping can be seen in Figure 2.6.1.

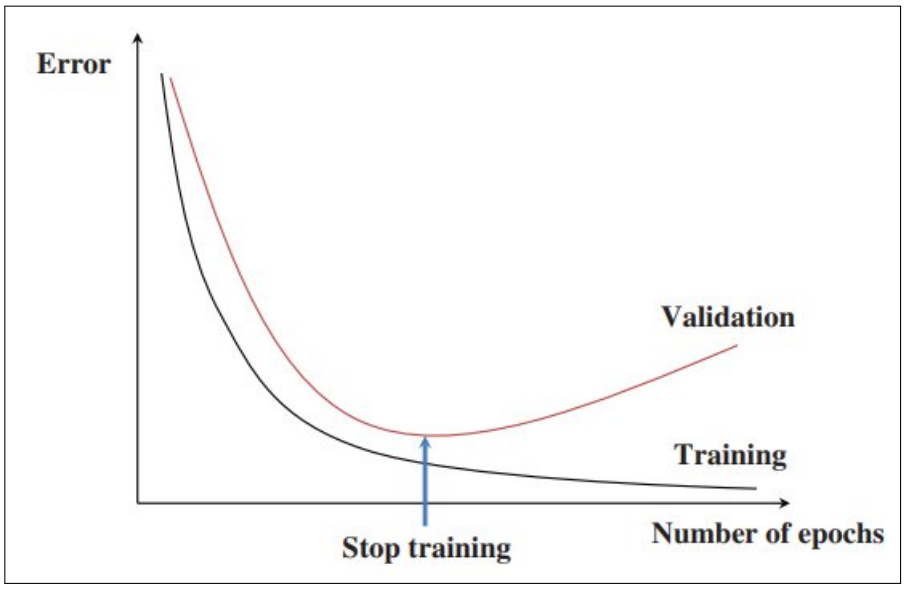

Figure 2.6.1: A graphical example of Early Stopping (Shin et al., 2016).

2.7 Performance Metrics

In neural network applications, model performance is assessed using three types of metrics, namely, training, validation and test metrics. Training metrics, typically rep-

resented by training loss curves, are computed on the training set and are primarily used to monitor convergence, diagnose instability during learning, and detect overfitting.

To conduct hyperparameter tuning and facilitate a direct comparison between candidate NN models, we compute standard regression error metrics on the validation set. The most commonly used metrics within the literature are the MAE and the MSE given by:

$$\text{MAE} = \frac{1}{N_{val}} \sum_{i=1}^{N_{val}} |\hat{y}_i - y_i|, \tag{2.32}$$

and

$$\text{MSE} = \frac{1}{N_{val}} \sum_{i=1}^{N_{val}} (\hat{y}_i - y_i)^2, \tag{2.33}$$

where N_{val} is the size of the validation dataset, y_i is the actual target value of the i^{th} observation and $\hat{y}_i$ is the predicted target value of the i^{th} observation. While the MAE and MSE are widely used regression metrics, they can be inadequate at comparing models when the target variables are expressed in different units or have different scales, as the numerical values are influenced by the units and magnitude of the variables. In the application found in Chapter 5, the target variables, latitude and longitude are measured in degrees, depth is measured in kilometres and magnitude is measured in magnitude units, meaning that direct comparison of MAE and MSE values across these variables is not meaningful without applying appropriate normalization measures. For this reason the NRMSE, based on standard deviation, defined in Equation (2.34), will also be utilized:

$$\text{NRMSE} = \frac{\sqrt{\text{MSE}}}{\sigma_y}, \tag{2.34}$$

where σ_y is the standard deviation of the true target values.

Predictive ability of the best model configurations is then determined on an independent test set by computing the Mean Absolute Error of Prediction (MAEP), Mean Squared Error of Prediction (MSEP) and Normalized Root Mean Squared Error of Prediction (NRMSEP). These measures are defined similar to Equations (2.32), (2.33) and (2.34), however, in this case, y_i represents the true target value of the i^{th} observation in the test set and $\hat{y}_i$ denotes the predicted target value of the i^{th} observation from the test set. Moreover, instead of N_{val}, the test set size N_{test} is used.

As will be seen in Section 5.4.5, when selecting the best hyperparameter combinations, validation MAE, MSE, and NRMSE for each target variable are compared separately, and an overall NRMSE, defined as the average of the per-target NRMSE values, is also used. Hyperparameter combinations producing the lowest validation errors are selected, and predictive ability is then assessed on the test set, where the model with the lowest test metrics is deemed the best-performing model.

Chapter 3

Convolutional Neural Networks

3.1 Introduction

Chapter 2 delved into the foundations of neural networks by discussing ANNs. This chapter focuses on an extension of ANNs known as Convolutional Neural Networks (CNNs). Introduced by LeCun (1989), CNNs are NNs that handle input data with a known grid-like topology. They are called *convolutional* neural networks because instead of using general matrix multiplication in forward propagation, they make use of a mathematical operation known as *convolution*. It is to be noted that the convolution used in NNs is not the same as the one used in other fields such as engineering and pure mathematics. More information on the convolution used in CNNs is given in Section 3.2.1.1. Apart from convolution, CNNs often make use of another operation referred to as *pooling*. This operation will be studied in greater detail in Section 3.2.2.

After briefly introducing CNNs, the rest of this chapter will be divided as follows. Section 3.2 discusses the basic architecture of a CNN, introducing convolutional layers, pooling layers and fully connected layers. In Sections 3.3 and 3.4 forward propagation and the UAT for CNNs are presented, respectively. Lastly, in Section 3.5, the CNN training procedure is discussed with focus on weight initialization and regularization methods. The motivation for presenting the theory behind CNNs in this chapter lies in their application in Chapter 5, where they will be employed to extract relevant information from waveform data and process the features. Consequently, most of the notation and examples presented here are tailored to their application in waveform data processing.

3.2 The Architecture of a Convolutional Neural Network

In Chapter 2 it was seen that an ANN is composed of an input layer, one or more hidden layers, and an output layer, where each layer applies matrix multiplications followed by non-linear activation functions to learn meaningful data representations. CNNs retain this fundamental structure but extend it with specialised layers, each serving a distinct purpose: *convolutional layers* extract local features, *pooling layers* perform spatial down-sampling, and *fully connected layers* conduct final inference. *Activation functions* within these layers again cater for the non-linear relationships between the explanatory variables (features) and the response variable. A schematic of the basic CNN architecture can be found in Figure 3.2.1. The remainder of this section reviews each layer in turn, detailing its role and its inner-workings.

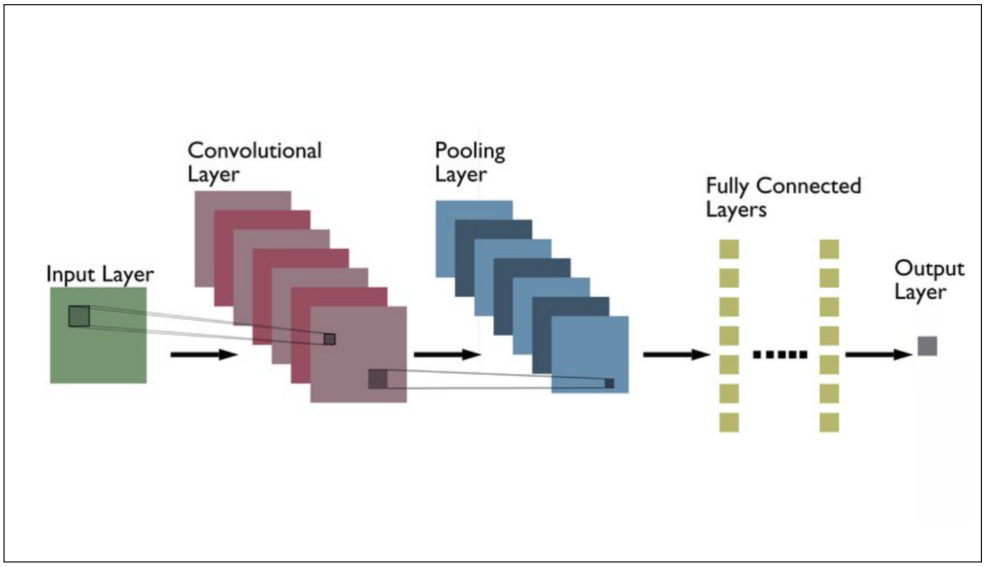

Figure 3.2.1: Simple CNN Structure (adapted from Kumar (2023)).

3.2.1 Convolutional Layers

This section delves into key components of convolutional layers, including the *convolution operator*, which enables CNNs to process data with a grid-like topology. It also discusses important concepts such as *sparse interactions*, *parameter sharing*, and *equivariant representations*, which enhance computational efficiency and generalization. Finally, the section details the hyperparameters of convolutional layers, namely, *kernel size*, *padding*, and *stride*, that influence feature extraction and output dimensionality.

3.2.1.1 The Convolution Operator

Convolutional layers are one of the principal parts of a CNN, as through them, the NN is able to process data with a grid like topology. This is carried out using a mathematical

operation referred to as convolution. In general, given two real-valued functions $x(\tilde{t})$ and $\omega(\tilde{t})$, where $\tilde{t}$ denotes the time or spatial index at which the convolution output is evaluated, convolution may be defined as follows in the continuous case:

$$z(\tilde{t}) = (x * \omega)(\tilde{t}) = \int x(b)\omega(\tilde{t} - b)db, \tag{3.1}$$

whereas, in the discrete case, convolution is defined as:

$$z(\tilde{t}) = (x * \omega)(\tilde{t}) = \sum_{b=-\infty}^{\infty} x(b)\omega(\tilde{t} - b). \tag{3.2}$$

Within the context of CNNs, x denotes the input and ω is referred to as the *kernel*. A set of kernels is then referred to as a *filter*. As shall be described in more detail shortly, the filter slides over the input data to calculate the convolution. The output $z(\tilde{t})$, resulting from this operation is termed the feature map. The feature map captures the kernel's response of each position, highlighting areas of the input that are activated by the filter.

The input can be viewed as a multidimensional array of data having dimension (N_s, N_t, D), denoted as $\mathbb{I}$, and the filter is a multidimensional array of dimension (K_{N_s}, K_{N_t}, D), denoted as $\mathbb{K}$, often of smaller size than the input, i.e., $K_{N_s} < N_s$ and $K_{N_t} < N_t$. The filter's role is to extract features from the input through performing a convolutional operation. These multidimensional arrays are referred to as *tensors*.

For example, the waveform input data that will be used in the application of Chapter 5 can be represented as a 3D tensor with dimensions defined by the *number of stations*, *number of time samples* and *seismic components*. Each element of the tensor stores a single time sample of the seismic event waveform recorded at a particular station and time step for the respective orthogonal seismic components (Z,N,E). For an explanation of a waveform, see Chapter 1. In a tensor, each time sample value is stored separately, allowing for independent processing. Thus, although the input might be a large, complex dataset, only elements that hold actual values are stored rather than storing values for the entire range of possible positions in an infinite space. So, although theoretically speaking, the kernel could be applied to every possible point in an infinite space, in practice we only sum over a finite number of array elements that contain the stored values.

Conventionally, NN libraries do not implement the mathematical convolution

operation directly, but instead use a similar operation called "cross-correlation". This should not be confused with the cross-correlation used in time series analysis, which has a different meaning. In many cases, this "cross-correlation" is applied over multiple axes simultaneously. For example, consider a waveform input $\mathbb{I}$ of shape (N_s, N_t, D), where N_s is the number of stations, N_t is the number of time steps per waveform, and D is the number of seismic components, which is three in our case. The resulting seismic-component feature map is computed simultaneously as:

$$\mathbb{Z}(i,j,d) = \sum_{n_s=1}^{K_{N_s}} \sum_{n_t=1}^{K_{N_t}} \mathbb{I}(i+n_s-1, j+n_t-1, d)\mathbb{K}(n_s, n_t, d). \tag{3.3}$$

Here $\mathbb{Z}(i,j,d)$ is the $(i,j,d)^{\text{th}}$ position of the output obtained after applying "cross-correlation", $\mathbb{I}(i+n_s-1, j+n_t-1, d)$ iterates over the input, where n_s and n_t shift the filter across the spatial and time dimension, respectively, and d indexes the seismic components. Furthermore, the filter's weights are indexed by $\mathbb{K}(n_s, n_t, d)$, where each index corresponds to a specific point in the kernel's spatial and time dimension (n_s, n_t), and depth dimension d. Lastly, K_{N_s} and K_{N_t} are the spatial and time dimension of the kernel. Since the convolution is performed independently for each seismic component, the final output feature map is obtained by aggregating the component-wise responses across the depth dimension:

$$\mathbb{Z}(i,j) = \sum_{d=1}^{D} \mathbb{Z}(i,j,d). \tag{3.4}$$

In order to better understand the "cross-correlation" function, a numerical example is considered. Suppose $\mathbb{I}$ is a $(3 \times 4 \times 3)$ input tensor and $\mathbb{K}$ is a $(2 \times 2 \times 3)$ filter. Each component is in its own "depth slice", so for each component $d \in \{1,2,3\}$, we define:

$$\mathbb{I}(i,j,1) = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}, \quad \mathbb{I}(i,j,2) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \end{bmatrix}, \quad \mathbb{I}(i,j,3) = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}, \tag{3.5}$$

and

$$\mathbb{K}(i,j,1) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbb{K}(i,j,2) = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad \mathbb{K}(i,j,3) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{3.6}$$

Applying the kernels in (3.6) to the top left position in the input tensors given by (3.5), the "cross-correlation" function for each seismic component is calculated as

follows:

$$\begin{aligned}
\mathbb{Z}(1,1,1) &= \sum_{n_s=1}^{2}\sum_{n_t=1}^{2} \mathbb{I}(1+n_s-1, 1+n_t-1, 1)\mathbb{K}(n_s, n_t, 1) \\
&= \sum_{n_s=1}^{2} [\mathbb{I}(n_s, 1, 1)\mathbb{K}(n_s, 1, 1) + \mathbb{I}(n_s, 2, 1)\mathbb{K}(n_s, 2, 1)] \\
&= \mathbb{I}(1,1,1)\mathbb{K}(1,1,1) + \mathbb{I}(1,2,1)\mathbb{K}(1,2,1) \\
&+ \mathbb{I}(2,1,1)\mathbb{K}(2,1,1) + \mathbb{I}(2,2,1)\mathbb{K}(2,2,1) \\
&= (1)(1) + (2)(0) + (5)(0) + (6)(1) \\
&= 7, \qquad (3.7)
\end{aligned}$$

$$\begin{aligned}
\mathbb{Z}(1,1,2) &= \sum_{n_s=1}^{2}\sum_{n_t=1}^{2} \mathbb{I}(1+n_s-1, 1+n_t-1, 2)\mathbb{K}(n_s, n_t, 2) \\
&= \sum_{n_s=1}^{2} [\mathbb{I}(n_s, 1, 2)\mathbb{K}(n_s, 1, 2) + \mathbb{I}(n_s, 2, 2)\mathbb{K}(n_s, 2, 2)] \\
&= \mathbb{I}(1,1,2)\mathbb{K}(1,1,2) + \mathbb{I}(1,2,2)\mathbb{K}(1,2,2) \\
&+ \mathbb{I}(2,1,2)\mathbb{K}(2,1,2) + \mathbb{I}(2,2,2)\mathbb{K}(2,2,2) \\
&= (1)(1) + (1)(1) + (2)(1) + (2)(1) \\
&= 6, \qquad (3.8)
\end{aligned}$$

and

$$\begin{aligned}
\mathbb{Z}(1,1,3) &= \sum_{n_s=1}^{2}\sum_{n_t=1}^{2} \mathbb{I}(1+n_s-1, 1+n_t-1, 3)\mathbb{K}(n_s, n_t, 3) \\
&= \sum_{n_s=1}^{2} [\mathbb{I}(n_s, 1, 3)\mathbb{K}(n_s, 1, 3) + \mathbb{I}(n_s, 2, 3)\mathbb{K}(n_s, 2, 3)] \\
&= \mathbb{I}(1,1,3)\mathbb{K}(1,1,3) + \mathbb{I}(1,2,3)\mathbb{K}(1,2,3) \\
&+ \mathbb{I}(2,1,3)\mathbb{K}(2,1,3) + \mathbb{I}(2,2,3)\mathbb{K}(2,2,3) \\
&= (0)(0) + (1)(1) + (1)(1) + (0)(0) \\
&= 2. \qquad (3.9)
\end{aligned}$$

Summing Equations (3.7), (3.8) and (3.9), we get the $(1,1)^{\text{th}}$ position of the feature map $\mathbb{Z}$:

$$\mathbb{Z}(1,1) = \mathbb{Z}(1,1,1) + \mathbb{Z}(1,1,2) + \mathbb{Z}(1,1,3) = 15. \qquad (3.10)$$

This process is then repeated on the input tensor for each kernel position, producing the following feature map:

$$\mathbb{Z} = \begin{bmatrix} 15 & 15 & 19 \\ 25 & 29 & 29 \end{bmatrix}. \tag{3.11}$$

where $\mathbb{Z}$ denotes the feature map obtained from applying a single filter to the input tensor using a cross-correlation operation, prior to the application of any non-linear activation function. Each entry of $\mathbb{Z}$ corresponds to the response of the filter at a specific kernel position.

3.2.1.2 Sparse Interactions, Parameter Sharing and Equivariant Representations

Convolution in a CNN is used in conjunction with three important mechanisms, namely, *sparse interactions*, *parameter sharing* and *equivariant representations*. The NNs seen in Chapter 2 make use of matrix multiplication through a matrix of weights where each individual weight describes the interaction between every input unit and every output unit. This suggests that each output unit interacts with each input unit. Unlike the conventional NNs seen in Chapter 2, CNNs usually have sparse connections, also known as *sparse weights* or *sparse connectivity*. Sparse connections are accomplished through making the kernel dimension smaller than the input size.

For example, suppose we are processing a seismic record of shape $(40, 4096, 3)$. A convolutional layer might use relatively small kernels, say of size (3×11), considering only 3 neighbouring stations and 11 successive time steps for each of the three seismic components. This subset would then contain just $3 \times 11 \times 3 = 99$ weights, whereas a fully connected layer would need to couple every station-time-component value to a single output unit resulting in $40 \times 4096 \times 3 \approx 5 \times 10^5$ weights. Despite this massive reduction in the number of weights that would need to be calculated, the main advantage of using kernels is that the network can still learn highly informative local features, such as sharp P-wave onsets, and then combine them in deeper layers to characterize the entire event. Thus, sparse connectivity, achieved through small kernels, results in fewer parameters being stored, reducing the CNN's memory requirements while improving how well the model learns from a limited amount of data. In practice, limiting the amount of connections between the input and output still makes it possible to attain good performance on the specific task at hand (Goodfellow et al., 2016). A graphical representation of sparse connectivity may be seen in Figures 3.2.2 and 3.2.3.

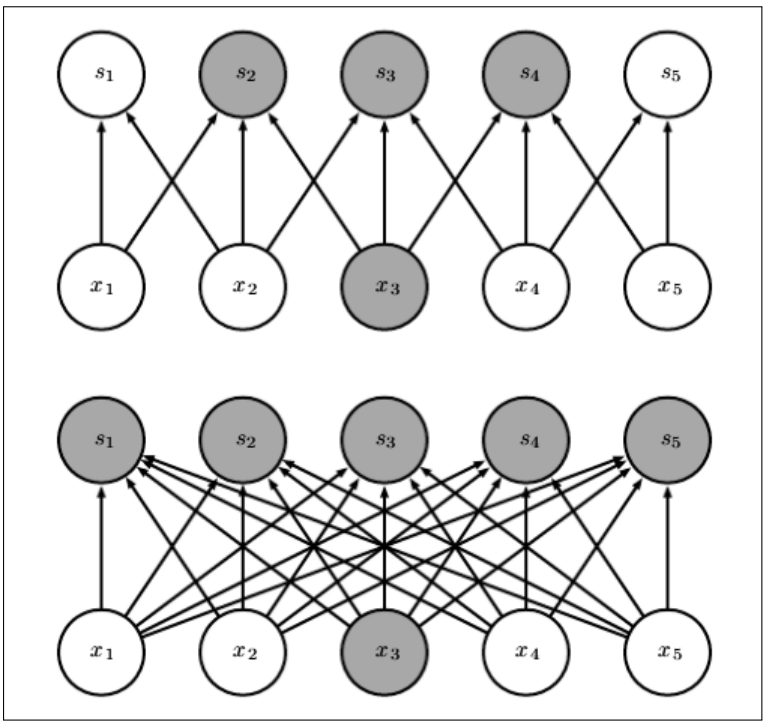

Figure 3.2.2: A representation of sparse connectivity as viewed from below (Goodfellow et al., 2016).

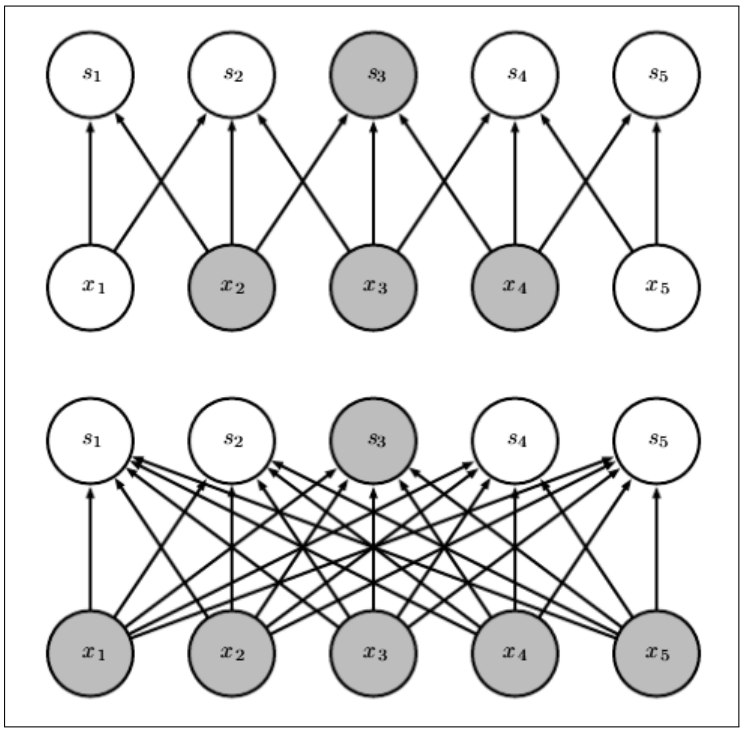

Figure 3.2.3: A representation of sparse connectivity as viewed from above (Goodfellow et al., 2016).

Figure 3.2.2 shows sparse connectivity viewed from below. Consider the input unit x_3 and the output units in $\boldsymbol{s} = (s_1, ..., s_5)'$ associated with x_3. The top half demonstrates the use of convolution while the bottom half demonstrates the use of matrix multiplication. Using convolution with a kernel of width 3 affects only 3 output units in $\boldsymbol{s}$. On the contrary, forming $\boldsymbol{s}$ through matrix multiplication, all output units

are affected by x_3 since the connectivity is no longer sparse.

Sparse connectivity viewed from above is then shown in Figure 3.2.3. Consider the output unit s_3 and the input units in $\boldsymbol{x} = (x_1, ..., x_5)'$ that affect this unit. These input units are referred to as the *receptive field* of s_3. As before, the top half demonstrates the use of convolution while the bottom half demonstrates the use of matrix multiplication. Only three inputs affect s_3 when $\boldsymbol{s}$ is generated using convolution with a filter of width 3, whereas when using matrix multiplication, all inputs affect s_3 as there is no sparse connectivity.

Now, in spite of having sparse connectivity, units in the deeper layers of a CNN can indirectly interact with a larger portion of the input, as illustrated in Figure 3.2.4. This enables the network to efficiently capture complex relationships among many variables by building these relationships from simple components, each of which represents only sparse interactions.

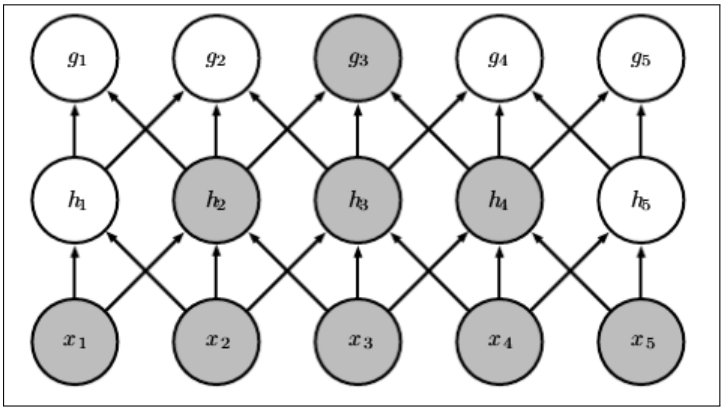

Figure 3.2.4: Illustration of how deeper layers in a convolutional network have larger receptive fields, allowing indirect connections to a broader portion of the input (Goodfellow et al., 2016).

As illustrated in Figure 3.2.4, the units in the deeper layers of a CNN have a larger receptive field than the units in the shallow layers. This effect is amplified when the CNN incorporates architectural features such as strided convolution or pooling, which will be discussed in Sections 3.2.1.3 and 3.2.2, respectively. So, despite the fact that *direct* connections in a CNN are sparse, units found in deeper layers may still be *indirectly* connected to most if not all of the input.

Now, certain NNs are designed in a way that the same parameter can be used for more than one part of the network. This is referred to as parameter sharing. In conventional NNs, each weight matrix element is multiplied by one input element and then never reconsidered. Thus, each weight is only used once to generate the layer output. In CNNs, however, each filter element is used at each position (n_s, n_t, d) of

the input. This is because the convolution operation within a CNN employs parameter sharing by learning one set of weights and applying them for every position. Although the forward propagation run time is not affected by this, the storage requirements of the NN are reduced since less parameters need to be calculated. Thus, when considering memory requirements, convolution is without a doubt more efficient than dense matrix multiplication (Goodfellow et al., 2016). A diagrammatic example of parameter sharing can be seen in Figure 3.2.5.

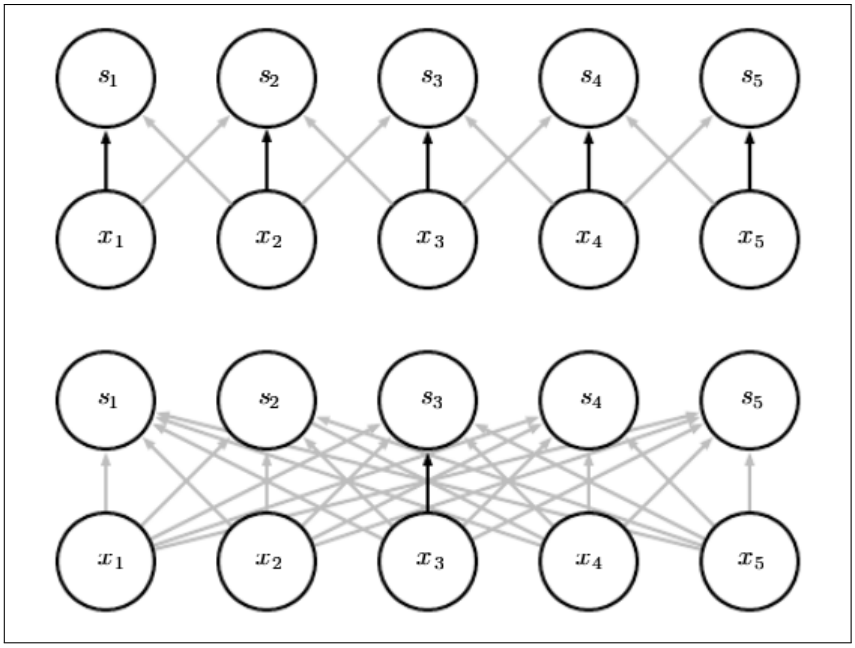

Figure 3.2.5: An example of parameter sharing (Goodfellow et al., 2016).

The top part of Figure 3.2.5 shows a CNN where the black arrows indicate connections that share a single parameter. The parameter in question corresponds to the central element of a 3-width kernel. As can be seen, because of parameter sharing, the same parameter is used for all locations of the input, allowing the NN to capture patterns that are the same spatially without requiring a separate parameter for each connection. On the other hand, the bottom part represents a fully connected model without the use of parameter sharing. In this case, the black arrow represents one use of the weight matrix's central element which connects a specific pair of nodes. In this NN, each pair of nodes are connected by their own unique parameter, so, the central parameter is only made use of once and is never revisited across different inputs.

The parameter sharing employed in convolutional layers causes the layer to have a property known as equivariance to transformation. A function is said to be equivariant if a change in the input results in the same change in the output. In particular, a function $\tilde{f}(x)$ is equivariant to a function $\tilde{g}$ if $\tilde{f}(\tilde{g}(x)) = \tilde{g}(\tilde{f}(x))$.

For instance, consider a three-component waveform input tensor $\mathbb{I}(n_s, n_t, d)$, where $n_s = 1, ..., N_s - 1$, $n_t = 1, ..., N_t - 1$ and $d = 1, 2, 3$. Let $\tilde{f}$ be a 2D convolution with a kernel $\mathbb{K}$ of fixed size (K_{N_s}, K_{N_t}) that is applied at every station-time location with shared weights. Define the time-shift operator as:

$$\tilde{g}_{\tilde{t}} : \mathbb{I} \mapsto (\tilde{g}_{\tilde{t}}\mathbb{I})(n_s, n_t, d) = \mathbb{I}(n_s, n_t - \tilde{t}, d),$$

i.e., a delay of $\tilde{t}$ time samples for all traces. Since the same kernel is used at every location,

$$\tilde{f}(\tilde{g}_{\tilde{t}}\mathbb{I})(n_s, n_t) = \sum_{i=0}^{K_{N_s}-1} \sum_{j=0}^{K_{N_t}-1} \sum_{d=1}^{3} \mathbb{K}(n_s, n_t, d)\mathbb{I}(n_s + i, n_t - \tilde{t} + j, d) = (\tilde{g}_{\tilde{t}}\tilde{f}(\mathbb{I}))(n_s, n_t).$$

Hence $\tilde{f} \circ \tilde{g}_{\tilde{t}} = \tilde{g}_{\tilde{t}} \circ \tilde{f}$, and so, the convolution is equivariant to time translation.

This equivariance to translation property is very useful since for example, the network needs to learn the shape of a P-wave onset only once, and it can recognise that onset anywhere in the waveform record without extra parameters. Nevertheless, it is to be noted that convolution is only equivariant to pure translations. Other transformations, such as time stretching or amplitude scaling, require additional mechanisms to be put into place.

3.2.1.3 Hyperparameters of Convolutional Layers

There are three hyperparameters that influence the output of a convolutional layer. These are known as *kernel size*, *padding* and *stride*. This section aims to briefly explain these terms.

The kernel size determines the dimension of the weight matrix (kernel) that slides over the positions of the input, directly impacting how detailed the detected features are. A larger kernel covers more input positions at once, so it removes more samples and produces a slightly smaller feature map. Applying a convolutional layer to an input tensor (N_s, N_t, D) trims $(K_{N_s} - 1)$ stations and $(K_{N_t} - 1)$ samples. Most networks use relatively small kernels, so the per-layer reduction is minor; however, stacking many such layers can still shrink the feature map substantially relative to the original input, resulting in a substantial loss of information. For instance, starting with 40×1024 samples, applying 64 successive convolutions with kernels of size (1×10) will shorten the time axis to 448 samples, almost 44% of the waveform trace. This issue can be handled using padding.

Padding works by adding dummy samples around the tensor to increase its effective size prior to sliding the kernel across it. Typically, the dummy samples are zero-valued. For example, suppose the "cross-correlation" function defined in Equation (3.3) is applied to a $(3 \times 3 \times 1)$ input tensor using a (2×2) kernel. This would result in a (2×2) output tensor. However, if the input tensor were to be padded, and its size increased to $(5 \times 5 \times 1)$, keeping the same kernel, the output tensor would have dimension $(4 \times 4 \times 1)$. This can be seen in Figure 3.2.6.

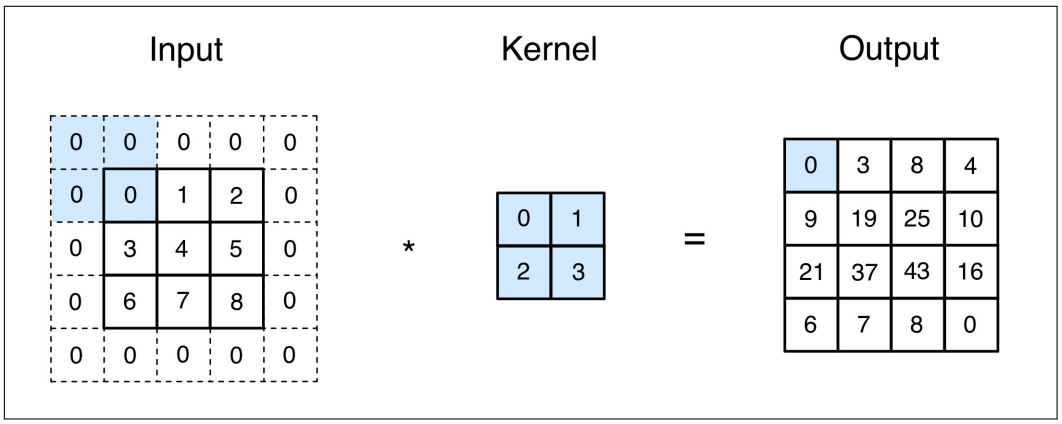

Figure 3.2.6: Two-dimensional cross-correlation with padding (A. Zhang et al., 2024).

In general, if p_{N_s} rows and p_{N_t} columns of padding are added, then the corresponding output would be of size:

$$(N_s - K_{N_s} + p_{N_s} + 1) \times (N_t - K_{N_t} + p_{N_t} + 1). \tag{3.12}$$

This suggests that the number of stations and number of time samples will be increased by p_{N_s} and p_{N_t}, respectively. In most cases, p_{N_s} and p_{N_t} are set as $p_{N_s} = K_{N_s}$ and $p_{N_t} = K_{N_t}$, this also applies for most Python libraries with the option *padding="same"*, so that the input layer and output layer would have the same dimension. This facilitates an easier way of knowing beforehand the output size of each CNN layer.

If K_{N_s} is odd, $\frac{p_{N_s}}{2}$ stations will be added before the first station and after the first station. If K_{N_s} is even, it is not as straight forward, however, one way to do it would be to pad $\lceil \frac{p_{N_s}}{2} \rceil$ stations before the first station and $\lfloor \frac{p_{N_s}}{2} \rfloor$ stations after the last station. This is done in a similar manner for the number of time samples. It is common to use kernels with odd values for K_{N_s} and K_{N_t} as it ensures that the dimensionality is preserved while padding with the same amount of stations before the first station and after the last station, as well as, the same amount of time samples before the start and after the end. If $K_{N_s} \neq K_{N_t}$, setting different padding numbers for p_{N_s} and p_{N_t} can

ensure that the input and output still have the same size.

Usually when the "cross-correlation" function is applied to the input tensor, the kernel starts at the first station and first time sample of the waveform tensor and slides over all possible indexes. In the examples that have been presented so far, the kernel moved only one element at a time. However, it can be the case that the kernel may be required to move more than one position at a time, skipping the intermediate positions. This procedure is especially useful when using large kernels since it captures a big portion of the input being analysed and is often employed when we want to boost the computational efficiency or down-sample (A. Zhang et al., 2024).

The number of stations/time steps traversed per slide is referred to as *stride*, denoted S_{N_s} and S_{N_t}, respectively. Sliding the kernel one position at a time means setting $S_{N_s} = 1$ and $S_{N_t} = 1$. Consider a waveform tensor of shape $(N_s, N_t, 3)$ with $N_s = 40$ stations and $N_t = 4096$ time samples. Suppose a convolutional layer with kernels having size (1×5) is applied with stride $(1, 2)$ and "same" padding. Along the station axis the stride is 1, so the kernel is evaluated at every station row, i.e no stations are skipped, and the output still has 40 rows. Along the time axis, the stride is 2, meaning the kernel moves two samples at a time. Consequently, only every second position is kept in the output. The length of the trace therefore drops from 4096 to 2048 samples. After this single layer, the tensor shape becomes $(40, 2048, F)$, where F is the number of kernels used. Since padding is set to "same", no extra shrinkage occurs at the start and end and so, the reduction is due purely to the stride.

In general, using a stride of S_{N_s} for the number of stations and a stride of S_{N_t} for the number of time samples would result in an output dimension of:

$$\left\lfloor \frac{(N_s - K_{N_s} + p_{N_s} + S_{N_s})}{S_{N_s}} \right\rfloor \times \left\lfloor \frac{(N_t - K_{N_t} + p_{N_t} + S_{N_t})}{S_{N_t}} \right\rfloor. \tag{3.13}$$

Setting $p_{N_s} = K_{N_s} - 1$ and $p_{N_t} = K_{N_t} - 1$, the output is simplified to:

$$\left\lfloor \frac{(N_s + S_{N_s} - 1)}{S_{N_s}} \right\rfloor \times \left\lfloor \frac{(N_t + S_{N_t} - 1)}{S_{N_t}} \right\rfloor. \tag{3.14}$$

Additionally, if the number of stations and number of time samples are exactly devisable by their respective strides, the output dimension would be:

$$\frac{N_s}{S_{N_s}} \times \frac{N_t}{S_{N_t}}. \tag{3.15}$$

In the application carried out in Chapter 5, padding is set to "same", and a stride of

(1,1) is used (Van den Ende & Ampuero, 2020; X. Zhang et al., 2022).

3.2.2 Pooling Layers

After applying activation functions in a CNN, pooling layers are used to further process the feature map by reducing its axes dimensions. First, a *pooling operation* is applied within each small subset of the input waveform components and aggregates information. Then the *downsampling operation* returns part of the aggregated data reducing the size of the waveform components and thus, decreasing the computational burden. For this reason, pooling layers are also known as downsampling layers. There are numerous pooling functions within the literature, however, the application presented in Chapter 5 will only make use of max pooling (Lee et al., 2017) and thus only this type of pooling will be presented here.

In max pooling, the filter slides over the activation map, selecting the maximum value from each region it covers. This produces a new activation map that returns only the most prominent activations from the original activations map. Given an activation map $\mathbb{O}$, and a pooling region of size $P_{N_s} \times P_{N_t}$, the max pooling operation is defined as:

$$\mathbb{P}(i,j) = \max_{\substack{0 \leq n_s < P_{N_s} \\ 0 \leq n_t < P_{N_t}}} \mathbb{O}(i \cdot P_{N_s} + n_s, j \cdot P_{N_t} + n_t). \tag{3.16}$$

Here $\mathbb{P}(i,j)$ is the pooling layer output at position (i,j). The activation map $\mathbb{O}$ represents the output obtained after applying a non-linear activation function to the corresponding feature map produced by the convolutional layer. An example of max pooling can be seen in Figure 3.2.7.

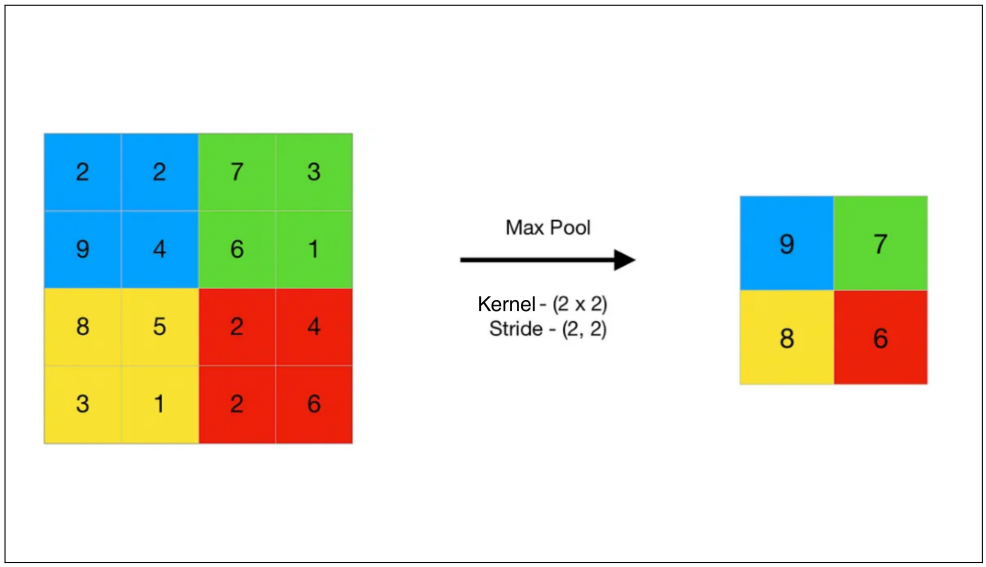

Figure 3.2.7: An example of max pooling using a (2×2) kernel and $(2,2)$ stride (Jain, A., 2024).

3.2.3 Fully Connected Layers

In a CNN, after the implementation of convolutional and pooling layers, a *fully connected layer* is introduced into the NN to further process the data and eventually generate the final required output. In short, the fully connected layers flatten the 2D spatial structure of the data into a 1D vector. Then, the features from the flattened feature maps are connected to each output by a learnable weight. These weights and biases are learned and updated throughout the training phases making the CNN adaptable to any task.

The amount of fully connected layers in a CNN is a hyperparameter that can affect then NN's performance and computational efficiency. Over-fitting may occur if there are too many fully connected layers. This is especially true if the NN has a considerable number of parameters relative to the size of the training data (Wu et al., 2017). On the contrary, if the number of fully connected layers is insufficient, the NN might struggle to learn complex patterns leading to a decrease in accuracy. Thus, ideally, the hyperparameter value chosen should guarantee a trade off between the NN's capacity to generalize without resulting in over-fitting.

3.3 Forward Propagation in CNNs

In the context of CNNs, forward propagation starts by taking an input and, through a series of operations and transformations, produces an output $\hat{y}$. While CNNs are conventionally applied to classification tasks, where $\hat{\boldsymbol{y}}$ typically represents a vector of probabilities corresponding to the different classes, in this work they are used to process the seismic data and produce a compact feature representation. In our case, $\hat{y}$ is a scalar representing the extracted feature value rather than a class label. Let $\mathbb{I}$ denote the input tensor for a seismic record of size $N_s \times N_t \times D$, where N_s represents the number of stations, N_t the number of time samples, and D the depth, i.e., the number of seismic components.

Forward propagation in a CNN begins in the convolutional layer, where for the l^{th} convolutional layer, using a filter $\mathbb{K}^{(l)}$ of dimension $K_{N_s}^{(l)} \times K_{N_t}^{(l)} \times D^{(l)}$, the outputted activation map $\mathbb{O}^{(l)}$ is calculated as follows:

$$\mathbb{O}^{(l)}(i,j) = \sigma^{(l)}\left[\sum_{n_s=1}^{K_{N_s}^{(l)}}\sum_{n_t=1}^{K_{N_t}^{(l)}}\sum_{d=1}^{D^{(l-1)}}\mathbb{O}^{(l-1)}(i+n_s, j+n_t, d)\cdot\mathbb{K}^{(l)}(n_s,n_t,d) + w_0^{(l)}\right]$$
$$= \sigma^{(l)}(\mathbb{Z}^{(l)}(i,j)) \tag{3.17}$$

where $\sigma^{(l)}$ and $w_0^{(l)}$ are the activation function and the bias pertaining to the l^{th} layer. If the convolutional layer is preceded by the input layer, then $\mathbb{O}^{(l-1)}$ is replaced by $\mathbb{I}$. Conversely, if the convolutional layer is preceded by a pooling layer, then $\mathbb{O}^{(l-1)}$ is substituted by $\mathbb{P}^{(l-1)}$.

Following the convolutional layer, a pooling layer that employs a pooling function like the max pooling function mentioned in Section 3.2.2 is introduced to reduce the dimension of the activation map. The output of the pooling layer for the l^{th} layer, $\mathbb{P}^{(l)}(i,j)$, is obtained as follows:

$$\mathbb{P}^{(l)}(i,j) = \underset{\substack{0\le n_s<P_{N_s}\\ 0\le n_t<P_{N_t}}}{pool}\ \mathbb{O}^{(l)}(i\cdot S_{N_s}+n_s, j\cdot S_{N_t}+n_t) \tag{3.18}$$

where *pool* represents a general pooling function. Furthermore, the conditions $0 \le n_s < P_{N_s}$ and $0 \le n_t < P_{N_t}$ define the bounds for iterating across the height n_s and n_t of the pooling window $P_{N_s} \times P_{N_t}$, making sure that each value within the indicated dimensions of the pooling region is covered. Lastly, $\mathbb{O}^{(l)}(i\cdot S_{N_s}+n_s, j\cdot S_{N_t}+n_t)$ indexes the particular elements in the l^{th} layer's output activation map that lie within the pooling region which are adjusted for n_s and n_s by the strides S_{N_s} and S_{N_t}.

As explained in Section 3.2.3 after pooling is implemented, the output of the l^{th} layer, $\mathbb{P}^{(l)}$, is flattened into a $(P_{N_s} \times P_{N_t} \times Q)$-vector, where Q is the number of activation maps:

$$\boldsymbol{V}^{(l)} = flatten(\mathbb{P}^{(l)}). \tag{3.19}$$

To better understand this, suppose after pooling we have an output of dimension $4 \times 4 \times 8$, which when multiplied together equals 128, thus, $\boldsymbol{V}^{(l)}$ is a 128-dimensional vector, i.e., $\boldsymbol{V}^{(l)} = (V_1^{(l)}, ..., V_{128}^{(l)})'$.

Another approach is to employ Global Max Pooling (GMP) in the l^{th} layer of a CNN to transform each of the $Q < D$ activation maps into a singular scalar value. GMP is carried out by taking the maximum value over the spatial dimensions of each

feature map:

$$GMP_q^{(l)} = \max_{0 \le i < P_{N_s}^{(l)}} \max_{0 \le j < P_{N_t}^{(l)}} \mathbb{P}_q^{(l)}(i,j), \tag{3.20}$$

where $GMP_q^{(l)}$ contains the most salient activation of the corresponding feature map. Furthermore, $P_{N_s}^{(l)}$ and $P_{N_t}^{(l)}$ are the number of stations and the number of time samples of the pooled feature maps of the l^{th} layer.

In the last forward propagation step, the flattened vector $\boldsymbol{V}^{(l)}$ or the $\boldsymbol{GMP}^{(l)} = (GMP_1^{(l)}, \ldots, GMP_Q^{(l)})'$ vector is passed onto the fully connected layer, ending at the final layer L producing an output $\hat{y}$:

$$\hat{y} = \sigma^{(L)}\Big(\boldsymbol{w}^{(L)} \cdot \boldsymbol{V}^{(L)} + w_0^{(L)}\Big) \tag{3.21}$$

where $\sigma^{(L)}$ is the activation function of the last layer L, and $\boldsymbol{w}^{(L)}$ and $w_0^{(L)}$ are the weight vector and bias of the last layer L, respectively. In addition to this, $\boldsymbol{V}^{(L)}$ represents the flattened vector from Equation (3.19), which can be alternatively replaced by the GMP vector in Equation (3.20).

3.4 The UAT for CNNs

In Section 2.2.3, the UAT for feedforward NNs was presented. This foundational result has also been extended to convolutional architectures. Zhou (2020) demonstrated that given an arbitrary precision, a CNN with enough layers can approximate any continuous function on a compact subset of $\mathbb{R}^{n_x}$. A concise statement of this result is presented next:

Theorem 2 UAT for CNNs

Given an arbitrary precision $\varepsilon > 0$ and any continuous function f on a compact subset T of $\mathbb{R}^{n_x}$, $\exists$ a CNN comprised of a finite sequence of convolutional layers, pooling layers and fully connected layers, using a non-polynomial activation function $\sigma(\cdot)$ such that the NN function $F(x)$ satisfies:

$$\sup_{x \in T} |F(x) - f(x)| < \varepsilon.$$

3.5 The CNN Training Procedure

Training CNNs follows the same foundational principles discussed in Chapter 2 for neural networks, including concepts such as gradient-based optimization and back-propagation. However, CNN training requires specific adaptations to account for the unique structural features of these networks. Thus, the aim of this section is to explore key aspects of CNN training, such as *weight initialization* and *regularization techniques*.

3.5.1 Weight Initialization

In CNNs with more than two convolutional layers, weight initialization plays a crucial role in determining both convergence and the network's ability to generalize. This process involves selecting the initial values for the network's weights *prior* to any learning, as these starting values directly influence the efficiency and stability of gradient-based optimization algorithms. Improper weight initialization can lead to challenges such as vanishing gradients, exploding gradients, or slow convergence (Glorot & Bengio, 2010). Furthermore, weight initialization is not always a straightforward process, as the setting up of these initial weights can vary depending on the network architecture.

The simplest approach is to initialize weights randomly within a small range. Another common method involves randomly selecting weight values from a zero-mean normal distribution with a variance close to zero. These methods were widely used in the early days of neural networks, when activation functions like the sigmoid function were prevalent. However, with sigmoid activation functions, poorly chosen initial weights can lead to gradient saturation, slowing or preventing learning. To address this, Xavier initialization (Glorot & Bengio, 2010) was invented.

Xavier initialization aims to ensure that the variance of activations remains consistent across all layers of a network. This method works particularly well for activation functions that are linear around the origin, such as the hyperbolic tangent activation function. This initialization method ensures that the expected variance of each layer's outputs matches its inputs, preventing activations and in turn gradients from vanishing or exploding during forward and backward passes.

In Xavier initialization, the weights are usually sampled from a Gaussian or uniform distribution:

$$w_{ii'}^{(l-1,l)} \sim N\left(0, \frac{2}{n_{in}+n_{out}}\right) \quad \text{or} \quad w_{ii'}^{(l-1,l)} \sim U\left(-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}}\right), \tag{3.22}$$

where $w_{ii'}^{(l-1,l)}$ is the weight joining the i^{th} neuron in the $(l-1)^{\text{th}}$ layer to the i'^{th} neuron in the l^{th} layer. Furthermore, n_{in} and n_{out} denote the number of units in the previous layer and the current layer, respectively.

Unfortunately, Xavier initialization does not cater for non-linear functions like the ReLU activation function. So, instead, He initialization (He et al., 2015) is preferred. This initialization method takes into account that the ReLU activation function and its variants output a zero for any negative input, which might lead to inactive units during training if not initialized correctly. In order to account for the one-sided nature of these functions, He initialization modifies the variance calculations by doubling the variance used in Xavier initialization. This is done to compensate for the reduced variance in the forward pass brought on by the non-linearity of ReLU activation functions. Thus, the weights are sampled from the normal and/or uniform distribution as follows:

$$w_{ii'}^{(l-1,l)} \sim N\left(0, \frac{2}{n_{in}}\right) \quad \text{or} \quad w_{ii'}^{(l-1,l)} \sim U\left(-\sqrt{\frac{6}{n_{in}}}, \sqrt{\frac{6}{n_{in}}}\right). \tag{3.23}$$

3.5.2 Regularization Techniques

As discussed in Section 2.6, regularization techniques for ANNs are a collection of algorithms designed to reduce the loss, mitigate overfitting/underfitting and help the model generalize better to unseen data. In the context of CNNs, regularization techniques are categorized according to the stage of the network where they are applied. The first category, referred to as *data augmentation*, includes algorithms that alter the input to a CNN. The second category, known as *internal changes*, encompasses algorithms that modify the internal parameters of the networks, such as kernel sizes and weights. The third category, called *label*, consists of algorithms that adjust the desired output. Data augmentation and label regularization algorithms will not be used in the application of this dissertation and so, will not be elaborated on further. The remainder of this section will discuss the regularization techniques used in Chapter 5.

Internal changes regularization methods are techniques that modify the internal parameters of a neural network, such as weights or kernel sizes, during training without explicitly altering the input data. One prominent example of this approach is the

conventional Dropout method utilized in ANNs. Dropout is a regularization technique used to reduce overfitting and improve generalization in a NN by randomly deactivating a subset of neurons during training. The extent to this stochastic deactivation is governed by the *dropout probability*, p_{drop}, which typically takes values between 0.2 and 0.5. In Chapter 5, p_{drop} is treated as a modifiable hyperparameter. For each neuron, a binary mask is sampled from a Bernoulli distribution, $Bernoulli(1 - p_{drop})$, such that neurons are retained with probability $(1 - p_{drop})$, and deactivated with probability p_{drop}. This stochastic mechanism discourages reliance on individual neurons and instead encourages the network to learn representations that remain informative even when subsets of neurons are absent, thereby improving stability and generalization to unseen data. Standard dropout can also be applied to CNNs. However, while standard dropout provides an effective regularization strategy, it also introduces certain limitations.

Specifically, standard dropout can increase the time complexity of backpropagation due to the need to account for deactivated neurons when updating weights. Moreover, dropping neurons entirely may result in a temporary reduction in model capacity, risking the loss of potentially valuable intermediate representations. So, over time, several variants of Dropout have been developed to enhance its functionality. Two variants that will be implemented together at different stages within the application presented in Chapter 5, are *Spatial Dropout* (Tompson et al., 2015) and *Gaussian Dropout* (Gal & Ghahramani, 2016).

Whilst conventional Dropout deactivates individual neurons, Spatial Dropout operates by randomly dropping entire activation maps (i.e., the seismic components). This approach is particularly effective in CNNs, as it preserves the spatial integrity of each feature map, ensuring that local patterns remain intact. At the same time, it promotes the development of complementary and diverse representations across different seismic channels, reducing the network's reliance on any single component and enhancing generalization (Tompson et al., 2015).

To also address the concerns mentioned above, Gaussian Dropout was proposed as a continuous and differentiable alternative to standard dropout (Karthik et al., 2022). Gaussian Dropout replaces the discrete Bernoulli mask used in standard dropout with multiplicative Gaussian noise. Instead of zeroing out individual activations, it perturbs each activation in the activation map by applying a randomly sampled scaling factor. Specifically, for each location (i, j) in the activation map $\mathbb{O}$, the activation $\mathbb{O}(i, j)$ is

multiplied by some noise variable $c(i,j)$, where $c(i,j) \sim \mathcal{N}\left(1, \frac{p_{drop}}{1-p_{drop}}\right)$ and p_{drop} is the dropout probability. This results in a noise-perturbed activation map $\tilde{\mathbb{O}}$, given by:

$$\tilde{\mathbb{O}}(i,j) = \mathbb{O}(i,j) \cdot c(i,j).$$

This formulation preserves the expected value of the activations (since $\mathbb{E}[c(i,j)] = 1$) while introducing variance that mimics the regularizing effect of standard dropout. The variance term $\frac{p_{drop}}{1-p_{drop}}$ increases with the dropout probability p_{drop}, allowing stronger regularization as needed.

Gaussian Dropout offers several advantages over standard dropout, both theoretically and practically. Empirical evidence suggests that Gaussian Dropout results in improved model performance and a lower time complexity relative to standard dropout (Karthik et al., 2022). Moreover, Gaussian Dropout has a principled Bayesian interpretation, where the noise can be viewed as modelling uncertainty in the network's activations (Gal & Ghahramani, 2016).

Chapter 4

The Graph Neural Network Framework

4.1 Introduction

Various real-world datasets can be naturally represented as a graph where entities correspond to nodes and their pairwise relations are expressed using edges. For instance, in the application in Chapter 5, stations are represented as nodes, and any relational dependencies between said stations are defined by edges. As a result, in Section 4.2, crucial concepts related to graphs will be outlined. Graph data is inherently different from grid structured data like images and sequences because graph data is irregular, unordered, and not fixed in size. This creates certain challenges for designing NN architectures capable of processing graphs effectively. This drove the development of a class of NNs that can operate effectively on graph data, referred to as GNNs.

In Chapter 5, two NN architectures will be explored. One architecture, referred to as the *edgeless graph architecture*, in which the graph structure contains only isolated nodes (i.e., no edges). The other architecture, called the *dynamic edges GNN*, has connections between stations generated dynamically according to a similarity criteria (see Section 4.4). Since the latter model is a GNN, the theoretical underpinnings of the GNN framework are detailed in this chapter. In Section 4.3, the basic GNN model, as well as how information is shared between nodes using message passing, is discussed. Additionally, Section 4.5 discusses the concept of universality, highlighting its relevance to the architectures considered in this dissertation.

4.2 Concepts Related to Graphs

Prior to outlining the GNN theoretical framework, this section introduces foundational concepts related to graphs. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined as a structure that consists

of a set of entities, called nodes (or vertices) $\mathcal{V}$, and a set of relationships, called edges $\mathcal{E}$, between these nodes.

Edges in a graph can be either *directed* or *undirected*. A directed edge $e \in \mathcal{E}$ has a source node $u \in \mathcal{V}$ and a destination node $v \in \mathcal{V}$, represented as $e = (u, v)$. In this case, information flows from u to v. In contrast, an undirected edge e does not have a designated source or destination node, and information flows in both directions. Mathematically, this is denoted by $(u, v) \in \mathcal{E} \Leftrightarrow (v, u) \in \mathcal{E}$. An example of a graph can be found in Figure 4.2.1.

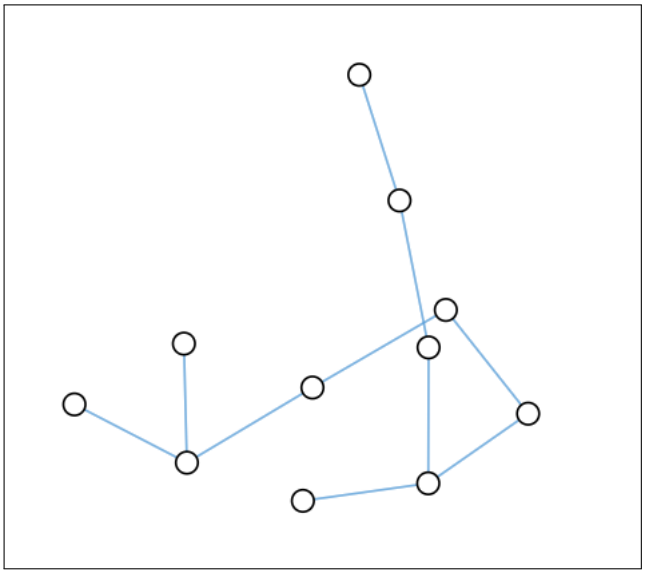

Figure 4.2.1: An example a graph (Sanchez-Lengeling et al., 2021).

Graphs are typically represented using an *adjacency matrix* $\mathbb{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$, where $|\mathcal{V}| = N$ is the number of nodes. The nodes in the graph are first ordered such that each node corresponds to a specific row and column in the adjacency matrix. The presence of edges is then encoded as elements of the matrix, where $\mathbb{A}[u, v] = 1$ if $(u, v) \in \mathcal{E}$, and $\mathbb{A}[u, v] = 0$ otherwise. The adjacency matrix $\mathbb{A}$ is symmetric if the graph contains only undirected edges. However, if the graph is directed, $\mathbb{A}$ will not necessarily be symmetric. In cases where the graph is used to represent the strength of association between two objects, the edges may be *weighted*, and the elements of $\mathbb{A}$ can take arbitrary real values rather than $\{0, 1\}$.

Most graphs have associated with them *attribute* or *feature* information. Typically, these are node-level attributes which are represented using a real-valued matrix $\mathbb{X} \in \mathbb{R}^{N \times n_x}$, assuming that the ordering of the nodes is consistent with the ordering in the adjacency matrix $\mathbb{A}$. Here n_x is the number of node-level features and in our case the attribute information consists of station locations and waveform features. In addi-

tion to node-level attributes, some graphs may also include real-valued edge features, which can encode properties such as edge weights or types. Furthermore, features can sometimes be associated with the entire graph, providing global information about the graph as a whole. An *attributed graph* is represented as $\mathcal{G}_{attrib} = (\mathcal{V}, \mathbb{A}, \mathbb{X})$.

Now, before delving into the basic GNN model, it is also worth mentioning that there are different kinds of learning problems that naturally arise on graphs. When applying NNs to graph structured data, the type of task depends on the level of the graph under consideration, where here the *level of the graph* refers to whether the prediction task concerns nodes, edges, or the entire graph.

Node level tasks focus on assigning values or labels to nodes, edge level tasks are concerned with the predictions of relations among pairs of nodes, and graph level tasks involve learning global properties of the entire graph. The application in Chapter 5 deals with a graph level regression task. In graph regression tasks, the learning set up involves a dataset of multiple different graphs, with the aim of predicting a target value for each graph independently. For instance in the application, seismic events are modelled as graphs in which the nodes correspond to seismic stations, each associated with features derived from waveform information and station location. The outputs of interest are then the source characteristics of the event, namely, latitude, longitude, depth and magnitude.

4.3 An Overview of Neural Message Passing and The Basic GNN Model

The term GNN refers to the general class of neural networks that operate on graphs through *neural message-passing*, where vector messages are exchanged between nodes and updated through neural networks (Gilmer et al., 2017). This section aims to explain how node embeddings, $\boldsymbol{z}_u$, for all $u \in \mathcal{V}$ are generated given an input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a set of node features $\mathbb{X} \in \mathbb{R}^{|\mathcal{V}| \times n_x}$, where n_x is the number of features, and $|\mathcal{V}| = N$ is the number of nodes. Additionally, this section also highlights how the message-passing mechanism fits into the basic GNN model and demonstrates how the neural message-passing approach can be simplified by incorporating self-loops into the input graph.

4.3.1 Neural Message Passing

In a GNN, graph information, such as node features and structural properties, is fed through message passing layers to construct node embeddings. A node embedding, denoted as $\boldsymbol{z}_u$, for a node $u \in \mathcal{V}$, is a vector representation that summarises the most relevant information about a node u's own features and from its neighbours found in $\mathcal{N}(u)$. The process involves gathering the current node information from its neighbours and combining it in various ways to obtain a new hidden embedding $\boldsymbol{h}_u$, $u \in \mathcal{V}$, which updates the node's features. This process is referred to as graph convolution, which can be thought of as an extension of the convolutional operations in CNNs, seen in Section 3.2.1.1, to irregular, graph-structured data.

Let L denote the total number of message-passing layers in the GNN, with $l = 0$ corresponding to the input layer and $l = L$ denoting the final layer. During the GNN message-passing procedure in layer l, $l = 0, ..., L-1$, a hidden embedding $\boldsymbol{h}_u^{(l)}$ is updated for each node $u \in \mathcal{V}$, based on information aggregated from its graph neighbourhood $\mathcal{N}(u)$. The message-passing update is mathematically defined as:

$$\boldsymbol{h}_u^{(l+1)} = \text{UPDATE}^{(l)}(\boldsymbol{h}_u^{(l)}, \text{AGGREGATE}^{(l)}(\{\boldsymbol{h}_v^{(l)}, \forall v \in \mathcal{N}(u)\})) \tag{4.1}$$

$$= \text{UPDATE}^{(l)}(\boldsymbol{h}_u^{(l)}, \boldsymbol{m}_{\mathcal{N}(u)}^{(l)}) \tag{4.2}$$

where UPDATE and AGGREGATE are differentiable functions and $\boldsymbol{m}_{\mathcal{N}(u)}^{(l)}$ denotes the "message" that is aggregated from node u's graph neighbourhood $\mathcal{N}(u)$ for the l^{th} layer, $l = 0, ..., L-1$. Aggregation is often implemented as:

$$\text{AGGREGATE}(M) := \mathcal{A}(\{f(i) | i \in M\}) \tag{4.3}$$

where M denotes the set of neighbour representations, f denotes some transformation $f : \mathbb{R} \to \mathbb{R}$, and $\mathcal{A}$ represents a permutation-invariant function, typically, *maximum*, *mean* or *summation*. In the simplest case, the transformation f is defined as $f(x) := \sigma(w \cdot x)$, where $w \in \mathbb{R}$ represents a learnable weight, and σ denotes a simple non-linear activation function. It is worth noting that certain applications may use a more sophisticated aggregation function, f, such as, an MLP (see Section 2.2.1 for an explanation of MLPs). In the application presented in Chapter 5, the *max pooling*

aggregation function is employed. For a node u, max pooling aggregation is defined as:

$$\boldsymbol{h}_u^{(l+1)} = \max_{v \in \mathcal{N}(u)} (\phi(\boldsymbol{h}_v^{(l)})), \tag{4.4}$$

where max is taken independently for each feature dimension. Here $\phi(\cdot)$ is a learnable function, typically an MLP. The simplest and most widely used update function is the MLP (Lutzeyer et al., 2022).

For each layer l, the set of embeddings of the nodes in u's graph neighbourhood $\mathcal{N}(u)$ are inputted into the AGGREGATE function and a message $\boldsymbol{m}_{\mathcal{N}(u)}^{(l)}$ is generated based on this aggregated neighbourhood information. Then, the message $\boldsymbol{m}_{\mathcal{N}(u)}^{(l)}$ is combined with the current embedding $\boldsymbol{h}_u^{(l)}$ of the node u via the update function UPDATE to generate the updated embedding $\boldsymbol{h}_u^{(l+1)}$. The initial embeddings at $l = 0$ are set to the input features for all the nodes i.e., $\boldsymbol{h}_u^{(0)} = \boldsymbol{x}_u$, $\forall u \in \mathcal{V}$. After N_l iterations of the GNN message passing, the output of the final layer is used to define the embeddings for each node i.e.,

$$\boldsymbol{z}_u = \boldsymbol{h}_u^{(L)}, \quad \forall u \in \mathcal{V}. \tag{4.5}$$

It is worth noting that GNNs defined in this way are permutation equivariant by design, since the AGGREGATE function takes as input a set of vectors $\{\boldsymbol{h}_v : v \in \mathcal{N}(u)\}$. Permutation equivariance means that if the ordering of the nodes (or neighbours) in the input graph is changed, the corresponding node representations produced by the network change in the same way, rather than being affected arbitrarily by the ordering. This property is essential for graph-structured data, as graphs do not possess a natural ordering of nodes or edges. Consequently, the learned node representations depend only on the graph structure and node features, and not on any arbitrary indexing of the nodes. An example of message-passing is presented in Section 4.3.2.

4.3.2 The Basic GNN Model

Up until now, the GNN framework has been discussed as a series of message-passing iterations using UPDATE and AGGREGATE functions which is relatively abstract. By giving concrete instantiations to these UPDATE and AGGREGATE functions, the abstract GNN framework defined in Equation (4.1) can be translated into a framework that can be implemented on real life datasets. The most basic GNN framework can be defined through the following simple GNN message passing:

$$\boldsymbol{h}_u^{(l+1)} = \sigma\left((\mathbb{W}_{self}^{(l,l+1)})^\intercal \boldsymbol{h}_u^{(l)} + (\mathbb{W}_{neigh}^{(l,l+1)})^\intercal \sum_{v\in\mathcal{N}(u)} \boldsymbol{h}_v^{(l)} + \boldsymbol{w}_0^{(l+1)}\right), \quad l = 0, ..., N_l, \tag{4.6}$$

where $\sum$ represented a general aggregation operator,$\mathbb{W}_{self}^{(l,l+1)}$ and $\mathbb{W}_{neigh}^{(l,l+1)}$ are $(n_{h_l} \times n_{h_{l+1}})$-matrices containing the weights joining layer l to layer $l+1$, associated with the self and neighbourhood nodes respectively, and $\boldsymbol{w}_0^{(l+1)} \in \mathbb{R}^{n_{h_l}}$ is a bias vector associated with the $(l+1)^{\text{th}}$ layer. Additionally, σ is an activation function, such as, tanh or ReLU, used to account for non-linearity in the model.

In addition to this, the basic GNN can be equivalently defined through the UPDATE and AGGREGATE functions mentioned in Section 4.3.1 as follows:

$$\begin{aligned} &\boldsymbol{m}_{\mathcal{N}(u)} = \sum_{v\in\mathcal{N}(u)} \boldsymbol{h}_v, \\ &\text{UPDATE}(\boldsymbol{h}_u, \boldsymbol{m}_{\mathcal{N}(u)}) = \sigma(\mathbb{W}_{self}\boldsymbol{h}_u + \mathbb{W}_{neigh}\boldsymbol{m}_{\mathcal{N}(u)}). \end{aligned} \tag{4.7}$$

Now, using an example, it will be illustrated how the input feature vector $\boldsymbol{x}_u$, $u \in \mathcal{V}$, is updated at each message passing layer, such that by the final message passing layer, the node embedding $\boldsymbol{z}_u$ contains the comprehensive, aggregated knowledge of the node and its neighbourhood.

Consider the input graph in Figure 4.3.1. To illustrate the message passing mechanism, we will perform two iterations of messaging passing on the input graph using Equation (4.7). Let the AGGREGATE function be a maximum function and let the UPDATE function be an MLP with a ReLU activation function. Furthermore, suppose $\mathbb{W}_{self}^{(0,1)} = \mathbb{W}_{neigh}^{(0,1)} = \mathbb{I}_2$, $\mathbb{W}_{self}^{(1,2)} = \begin{bmatrix} 1 & 1 \\ -1 & 2 \end{bmatrix}$ and $\mathbb{W}_{neigh}^{(1,2)} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$.

To update the input feature vector $\boldsymbol{x}_2$ presented in Figure 4.3.1, information from its neighbouring nodes $\mathcal{N}(2) = \{1, 3\}$, found in $\boldsymbol{x}_1$ and $\boldsymbol{x}_3$ is aggregated:

$$\boldsymbol{m}_{\mathcal{N}(2)}^{(1)} = \max\{\boldsymbol{h}_1^{(0)}, \boldsymbol{h}_3^{(0)}\} = \max\{[1, 0]', [3, 1]'\} = [3, 1]'. \tag{4.8}$$

The current node's features in $\boldsymbol{x}_2$ are then combined with the aggregated data from Nodes 1 and 3, to form the hidden embedding $\boldsymbol{h}_2^{(1)}$, which reflects both the node's own

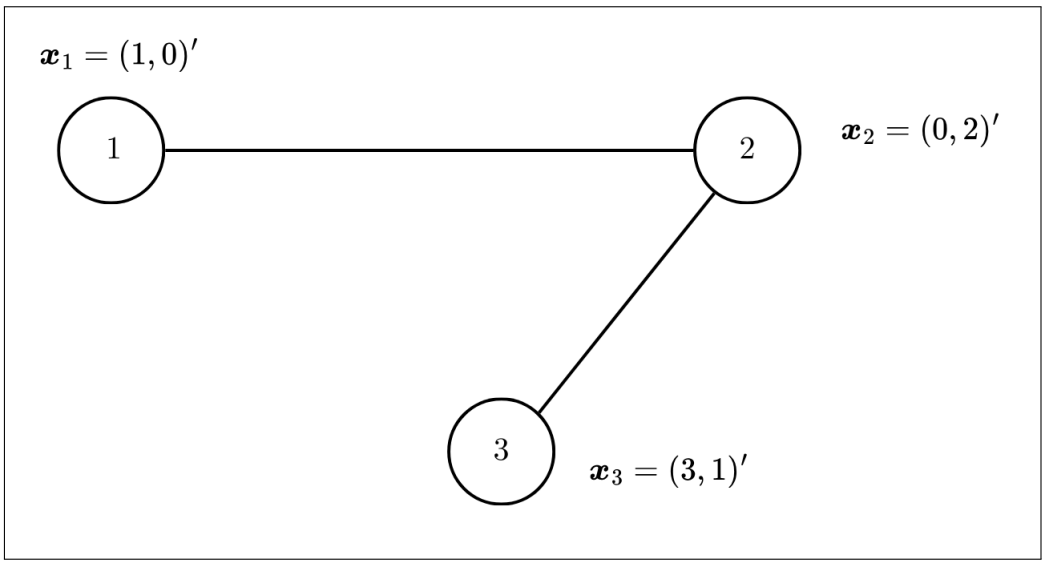

Figure 4.3.1: Message Passing Example: Input Graph.

features and the neighbours' information:

$$\begin{aligned}
\boldsymbol{h}_2^{(1)} &= \sigma\left(\mathbb{W}_{self}^{(0,1)}\boldsymbol{h}_2^{(0)} + \mathbb{W}_{neigh}^{(0,1)}\boldsymbol{m}_{\mathcal{N}(2)}^{(1)}\right) \\
&= \text{ReLU}\left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 0 \\ 2 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 3 \\ 1 \end{bmatrix}\right) \\
&= \text{ReLU}\left(\begin{bmatrix} 0 \\ 2 \end{bmatrix} + \begin{bmatrix} 3 \\ 1 \end{bmatrix}\right) = \text{ReLU}([3, 3]') = [3, 3]'.
\end{aligned} \tag{4.9}$$

Performing an analogous procedure to Nodes 1 and 3, the new graph, shown in Figure 4.3.2, is then obtained.

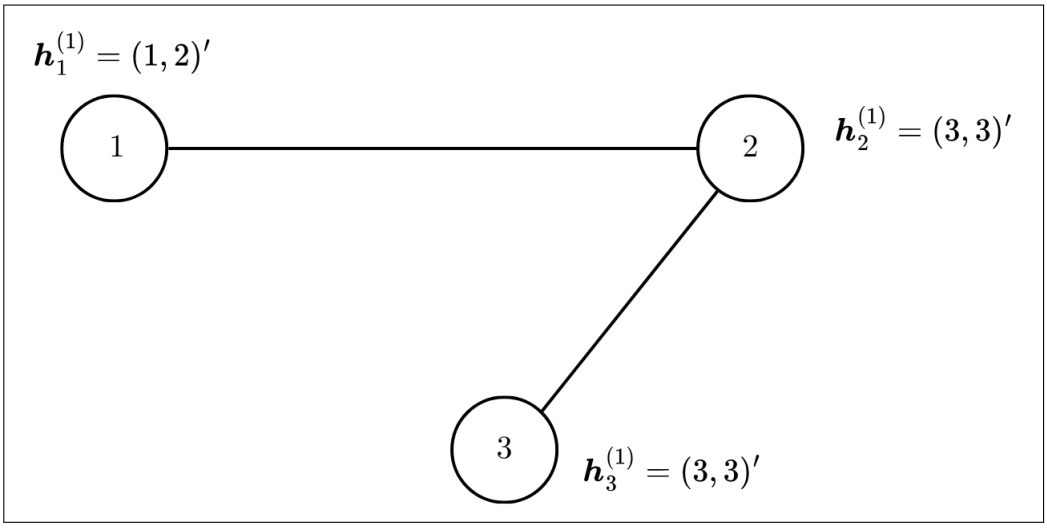

Figure 4.3.2: Message Passing Example: Updated Graph (after the first Message Passing layer).

After one message-passing step, each node only aggregates information from its immediate neighbours. Consequently, Nodes 1 and 3 are unaware of each other. However, this will change after performing a second message-passing step. Using Node 1, aggregating and updating is carried out as follows:

$$\boldsymbol{m}_{\mathcal{N}(1)}^{(2)} = \max\{\boldsymbol{h}_2^{(1)}\} = \max\{[3,3]'\} = [3,3]', \tag{4.10}$$

$$\begin{aligned}
\boldsymbol{h}_1^{(2)} &= \sigma\left(\mathbb{W}_{self}^{(1,2)}\boldsymbol{h}_1^{(1)} + \mathbb{W}_{neigh}^{(1,2)}\boldsymbol{m}_{\mathcal{N}(1)}^{(2)}\right) \\
&= \text{ReLU}\left(\begin{bmatrix} 1 & 1 \\ -1 & 2 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} 3 \\ 3 \end{bmatrix}\right) \\
&= \text{ReLU}\left(\begin{bmatrix} 3 \\ 3 \end{bmatrix} + \begin{bmatrix} 6 \\ 6 \end{bmatrix}\right) = \text{ReLU}([9,9]') = [9,9]'.
\end{aligned} \tag{4.11}$$

Performing an analogous procedure to Nodes 2 and 3, the new graph, shown in Figure 4.3.3, is obtained. This process of aggregating and updating is carried out iteratively over multiple layers, eventually producing the final node embeddings $\boldsymbol{z}_1, \boldsymbol{z}_2$ and $\boldsymbol{z}_3$.

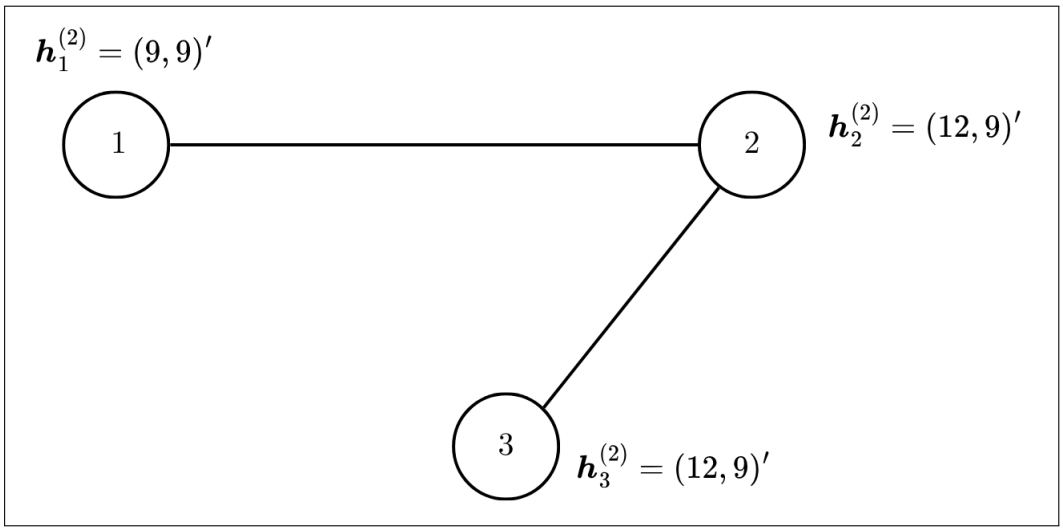

Figure 4.3.3: Message Passing Example: Updated Graph (after the second Message Passing layer).

The basic GNN model outlined in Equation (4.7) can achieve great performance while being easy to understand (Hamilton, 2020). The basic GNN can be generalized in various ways, but generalizations will not be needed for the purpose of the application

tackled in this dissertation. The GNN used in the application is relatively simple, in the sense that it does not need to employ generalized neighbourhood aggregation functions or sophisticated update mechanism.

In dynamic GNNs, the neighbourhood of a node u, $\mathcal{N}(u)$, is not fixed across layers. Instead, it may be dynamically "redefined" at each layer based on the "evolving" node representations $\boldsymbol{h}_u^{(l)}$. For instance, edge connections can be recalculated using feature similarity, or even learned using NN that predict edge scores or attention coefficients. This enables the GNN to adapt its message-passing structure as learning progresses. For example, as shall be seen, some models in the application in Chapter 5 generate edges dynamically based on feature similarity using Euclidean distances.

The core message-passing operations in Equation (4.6) are defined at the node-level. It is worth noting that many GNNs can also be defined using graph-level equations. For a basic GNN, the graph-level definition of the model can be written as follows:

$$\mathbb{H}^{(l+1)} = \sigma(\mathbb{A}\mathbb{H}^{(l)}\mathbb{W}_{neigh}^{(l+1)} + \mathbb{H}^{(l)}\mathbb{W}_{self}^{(l+1)}), \tag{4.12}$$

where $\mathbb{H}^{(l+1)}$ is a $(N \times n_{h_l})$-matrix of node representations at the $(l+1)^{\text{th}}$ layer in the GNN, with each node corresponding to a row in the matrix and $\mathbb{A}$ denotes the graph adjacency matrix. Additionally, the bias term has been omitted for notational simplicity. This graph-level representation highlights how many GNNs can be efficiently implemented through a small number of sparse matrix operations.

Through successive layers of message passing, the model produces final node embeddings $\boldsymbol{h}_u^L$, represented as a row in $\mathbb{H}^{(L)}$. Subsequently, the set of final node embeddings is combined using a READOUT function to obtain a single embedding for the entire graph. Mathematically, this can be expressed as:

$$\boldsymbol{z}_{\mathcal{G}} = \text{READOUT}(\{\boldsymbol{z}_u : u \in \mathcal{V}\}), \tag{4.13}$$

where $\boldsymbol{z}_{\mathcal{G}}$ is the graph embedding, summarizing the information from all nodes. The readout function is typically selected in such a way that it is equivariant, with common choices being sum pooling, average pooling, max pooling or attention-based pooling. Max pooling was explained previously in Section 3.2.2, the other types of pooling will not be used in the application, and thus, no further detail will be provided. Finally, the graph embedding $\boldsymbol{z}_{\mathcal{G}}$ is fed through an output function, typically an MLP or a linear layer, to generate the predicted properties of a graph, $\hat{\boldsymbol{y}}$, where in our case,

$\hat{\boldsymbol{y}} = (latitude, longitude, depth, magnitude)$.

4.3.3 Message Passing with Self-Loops

To avoid defining the UPDATE function in the neural message passing explicitly, and thus, simplifying the procedure, it is convention to include self-loops to the input graph and omit the explicit update step. Using this approach, message passing is then simply defined as:

$$\boldsymbol{h}_u^{(l+1)} = \text{AGGREGATE}(\{\boldsymbol{h}_u^{(l)}, \forall v \in \mathcal{N}(u) \cup \{u\}\}) \tag{4.14}$$

where now the aggregation is taken over the set $\mathcal{N}(u) \cup \{u\}$ i.e., the node's neighbours as well as the node itself. This formulation represents the nodal update rule.

At the matrix level, adding self-loops to a basic GNN is the same as sharing parameters between the $\mathbb{W}_{self}$ and $\mathbb{W}_{neigh}$ matrices, resulting in the following graph-level update:

$$\mathbb{H}^{(l+1)} = \sigma\left((\mathbb{A} + \mathbb{I}_N)\mathbb{H}^{(l)}\mathbb{W}^{(l+1)}\right) \tag{4.15}$$

where $\mathbb{H}^{(l+1)}$ is a $(N \times n_{h_l})$-matrix of node representations at the $(l+1)^{\text{th}}$ layer in the GNN, with each node corresponding to a row in the matrix and $\mathbb{A}$ denotes the graph adjacency matrix and $\mathbb{I}_N$ is a $(N \times N)$-identity matrix. Throughout the rest of this dissertation, the method discussed in this section will be referred to as the *self-loop* GNN approach. The dynamic edges GNN in Chapter 5 is defined in terms of self loops.

4.4 Dynamic Edge Generation and Feature Fusion

Some of the GNN models that will be fitted in Chapter 5 leverage dynamically generated edges to build flexible and informative connections between nodes. In these models, the seismic station network is represented as a graph in which each node corresponds to a seismic station, and edges between stations are generated automatically based on various criteria. In particular, the edges will be generated using geographic locations of the stations and similarities in their waveform features. This EdgeConv-based (see Section 1.4.3) approach is inspired by the work of X. Zhang et al. (2022). Through iterative steps of edge generation and graph convolution, the model progressively expands the receptive field of each node, allowing it to integrate information from an increasingly larger portion of the network. This enables the model to fuse features from multiple stations and capture higher-order spatio-temporal patterns in the recorded wavefield across the seismic network. Consequently, this section aims to

introduce dynamic edge generation and feature fusion procedure.

Edge Generation

Edges are dynamically generated by linking every station with its k-nearest neighbours according to their similarity. Here k is a modifiable hyperparameter and the similarity is determined based on two criteria, namely, *geographic distance* and *feature similarity*. Determining similarity based on geographic distance is intuitive, since adjacent stations often record related signals as a result of similar wave paths. Moreover, seismic events are more likely to be mutually recorded by stations in near proximity, particularly in the case of small magnitude earthquakes. Nevertheless, the same seismic event can be recorded by distant stations in a large area, and so, waveform and feature similarity provide a complimentary approach to geographic distance. The difference between waveform/additional feature information between station u and v is determined directly using l^2 distance.

Not taking into account the seismic channel and batch dimensions, the feature matrix for neighbour selection is assumed to be $\mathbb{X}^{N \times n_x}$, where N is the number of stations and n_x is the number of features. When the criterion for generating edges is geographic distance (i.e., the features are the station coordinates), $n_x = 2$. The process of choosing k-nearest neighbours can be summarized using the following two steps. Given feature vectors $\boldsymbol{x}_u \in \mathbb{R}^{n_x}$ and $\boldsymbol{x}_v \in \mathbb{R}^{n_x}$ for stations u and v, respectively, the pairwise distance of stations u and v is computed using:

$$S(u, v) = ||\boldsymbol{x}_u - \boldsymbol{x}_v||_2, \quad 1 \leq u, v, \leq N. \tag{4.16}$$

Evaluating $S(u, v)$ for all station pairs then yields the pairwise distance matrix $\mathbb{S} \in \mathbb{R}^{N \times N}$, where entries are given by:

$$\mathbb{S}_{uv} = S(u, v). \tag{4.17}$$

After this, the k-nearest neighbours $\mathcal{N} \in \mathbb{R}^{N \times k}$ are obtained by sorting each row of $\mathbb{S}$:

$$\mathcal{N}(u) = \textit{smallest-}k(\mathbb{S}(u)), \quad 1 \leq u \leq N. \tag{4.18}$$

Feature Update

Having generated the edges, the features of each station are updated using:

$$\tilde{\boldsymbol{x}}_u = maxpool_{v \in \mathcal{N}_{dist}(u)}(g(\boldsymbol{x}_u - \boldsymbol{x}_v) + f(\boldsymbol{x}_u)) + maxpool_{v' \in \mathcal{N}_{sim}(u)}(g'(\boldsymbol{x}_u - \boldsymbol{x}_{v'}) + f'(\boldsymbol{x}_u)), \tag{4.19}$$

where $\boldsymbol{x}_u$, $\boldsymbol{x}_v$ and $\boldsymbol{x}_{v'}$ are the feature vectors associated with stations u, v and v', respectively. Furthermore, v is a neighbour of u based on geographic distance and v' is a neighbour of u based on feature similarity, determined from the previous step. Additionally, $g(\cdot), f(\cdot), g'(\cdot)$ and $f'(\cdot)$ are trainable fully connected NNs and $\tilde{\boldsymbol{x}}_u$ is the updated feature vector of station u. Lastly, max-pooling is performed along the constructed edges to combine information from the k-nearest neighbours of u, in order for each station to be once more associated with a single feature vector.

It is worth noting that the update step is asymmetric with respect to stations u and v. This is because, the message passed from station v to station u is dependent on the feature difference $\boldsymbol{x}_u - \boldsymbol{x}_v$ and on the transformations applied to $\boldsymbol{x}_u$. As a result, the update of station u is not equivalent to the update that would be obtained by switching the roles of u and v. This asymmetry is intentional and reflects the fact that seismic stations may record the same event with different signal quality. Thus, information from one station can be informative for another without the opposite necessarily being true.

4.5 Universality of the Proposed Neural Networks

The universal approximation capabilities of feedforward NNs and CNNs have been discussed at length in Sections 2.2.3 and 3.4, whereby it was seen that feedforward NNs and CNNs are capable of representing any continuous function, as long as the activation function satisfies a set of conditions. This section will now delve into the universal approximation capabilities of the NNs that will be used in Chapter 5.

Universality as a Set Function Approximator

As will be seen later on in Sections 5.5.1 and 5.6.1, the last stage of the models implements a permutation-invariant max pooling operator across all stations, followed by an MLP. Theoretical results found in the NN literature show that such architectures are capable of universal approximations of continuous functions defined on sets. For instance, Zaheer et al. (2017) proved that any continuous permutation-invariant function (i.e., its output does not change when the order of its input elements is rearranged), f

acting on a finite set $\{x_1, ..., x_n\}$ can be decomposed as:

$$f(\{x_i\}) = \theta\left(\sum_{i=1}^{n} \phi(x_i)\right), \tag{4.20}$$

for suitable transformations, typically MLPs, θ and ϕ. This result establishes universality when the aggregation is conducted using a sum operator. Following this, Qi et al. (2017) showed in the *PointNet* NN architecture that max pooling can also achieve universality for continuous set functions, under the assumptions of compactness and continuity with respect to the Hausdorff distance:

$$f(\{x_i\}) = \theta\left(\max_i \phi(x_i)\right), \tag{4.21}$$

where θ and ϕ are as defined before. This result suggests that max-based aggregation, followed by a sufficiently expressive MLP, maintains the ability to approximate any continuous permutation invariant mapping from a set of inputs to an output. The proposed architectures implement masked max pooling, whereby a binary mask is used to exclude padded or missing entries from the max pooling operation, and so, they inherit this universality guarantee at the set function level.

Universality as a GNN

Although the dynamic edgess GNN architecture is a universal set function approximator, its expressive ability from a graph-theoretic perspective is a bit more restricted. The expressive power of Message-Passing Neural Networks (MPNNs) is at most as strong as the 1-Weisfeiler-Leman (WL) graph isomorphism test (Morris et al., 2019; Xu et al., 2019). A comprehensive overview of the WL-Algorithm can be found in Appendix D. This means that no standard MPNN can act as a universal approximator for all possible graph invariant functions.

Thus, within the MPNN framework, the choice of aggregation operator plays a crucial role. Xu et al. (2019) showed that sum aggregation, combined with injective update functions, achieves the full discriminative power of the 1-WL test. Max and mean aggregators are in general weaker, as they might not be able to capture certain multi-set distinctions that sum aggregation preserves. The dynamic edges GNN architecture uses max aggregation in its message passing layers. Consequently, we need to acknowledge the fact that its universality is bounded by the 1-WL limitation of MPNNs, as well as the reduced discriminating power of max pooling in comparison with sum.

However, taking everything into account, while the dynamic edges GNN architecture cannot be regarded as a universal approximator for arbitrary graph functions, it still benefits from strong universality guarantees as a permutation-invariant set function approximator. Thus, the dynamic edges GNN can still be considered to provide sound theoretical foundation for its use in modelling seismic station networks, where invariance to station ordering is a natural and necessary property.

Chapter 5

Application and Results

5.1 Introduction

After having outlined the core principles of ANNs, CNNs and GNNs in Chapters 2-4, a practical application of GNNs to a seismic source characterization task focused on the central Mediterranean region is now presented. Two primary model architectures are explored: an edgeless graph architecture and a GNN with dynamically constructed edges based on criteria defined in Section 4.4. An experimental approach is adopted, involving systematic adjustments to both neural network parameters and architectural components to optimize performance. Given the geographic spread of seismic events, initial observations suggest the presence of potential subgroups. Consequently, clustering techniques are applied to uncover groupings in the dataset, under the hypothesis that improved homogeneity may facilitate more accurate source prediction. All implementation is carried out in Python.

The chapter is structured as follows. Section 5.2 describes the computational setup and Python environment, while Section 5.3 details the data acquisition and preprocessing pipeline. Section 5.4 outlines the model standards, training procedure, and evaluation approaches. The design and execution of the various experiments, along with preliminary results, are discussed in Sections 5.5 and 5.6. In Section 5.7, the clustering-based approach is introduced, and all experiments are replicated on the clustered data. Then, in Section 5.8, we compare the top two models from Sections 5.5 and 5.6, and the top two from Sections 5.7.1 and 5.7.2. The comparison is based on validation metrics, loss plots, and prediction plots. We also implement an ensemble modelling approach to assess whether ensembling improves predictive accuracy. Finally, the best two models from Sections 5.5 and 5.6 are fitted on a held-out test

dataset, followed by some concluding remarks.

5.2 Computational Set Up

This section outlines the computational set up used to carry out the application. All experiments were conducted on a local machine, with the relevant hardware specifications detailed shortly. In addition, a summary of the Python environment, including the tools, libraries and modules utilized, is provided. This information is necessary to ensure reproducibility of results and transparency of implementation.

5.2.1 Hardware Specifications of Local Machine

The computational experiments were executed on a local system equipped with an 11th Gen Intel® Core™ i9-11900K CPU running at 3.50 GHz with 8 cores and 16 threads, providing robust parallel processing capabilities. An NVIDIA GeForce RTX 3090 served as the primary GPU, with approximately 4.29 onboard memory. The system was fitted with 64 GB of Patriot memory (PDP Systems) RAM, operating at a speed of 3200 MHz. The system ran on a 64-bit installation of Windows 10 Enterprise.

5.2.2 Python Environment

The Python environment consisted of several key tools, libraries and modules to facilitate the construction and execution of the neural networks of this application. Downloading and processing of seismic data was performed using the `ObsPy` library while visualization of said data was achieved through the `matplotlib` and `cartopy` libraries. The main library utilized to build the neural networks and train the models was `TensorFlow`. Its integrated `Keras` API was used to design both sequential and functional model architectures, allowing for flexible and efficient implementation of the deep learning models. The fine tuning and optimization of models was facilitated through advanced `TensorFlow` modules, such as custom callbacks and `Keras` layers. Any data manipulation and numerical operations were executed using libraries such as `NumPy` and `Pandas`. Additionally, file and directory management was handled using the `os` library, while the `pickle` module was employed for serialising and deserialising Python object structures. Lastly, the functions `KMeans` and `DBSCAN` from the `sklearn.cluster` module, and the module `hdbscan` were used to carry out K-means, DBSCAN and HDBSCAN clustering.

5.3 The Dataset and Pre-processing

As discussed in Section 1.3, the aim of this dissertation is to establish a statistical model to characterize the seismic source of earthquakes in the central Mediterranean region by predicting the latitude, longitude, depth and magnitude of each seismic event. In order to achieve this, the models will be trained and validated on a central Mediterranean region dataset comprising waveform (see Section 1.1) and earthquake catalogue information collected from the Istituto Nazionale di Geofisica e Vulcanologia (INGV) data centre from the seismic networks IV, MN and ML. The dataset contains seismic events for the period January 2013 to December 2024 within a geographic subset from 34° to 39° latitude and 11° to 16.5° longitude, a depth range of 0-30km, and a magnitude range of [3,5.5]. Seismic events having magnitude of at least 3 on the Richter scale were chosen for this study, as smaller events are often too weak to be considered reliable earthquake detections. The largest recorded magnitude in this region during this period is 5.5. In total, the data set consists of 685 events recorded by 55 broadband seismic stations. Following Dr Agius' advice, earthquakes that occurred before 2013 were not considered as they might not necessarily meet the required standards of reliability of measurement. This is because, prior to 2013, fewer high-quality broadband seismic stations were available, resulting in good-quality data being more sparsely distributed. In contrast, the more recent dataset used in this study is more convenient, consistent, and homogeneous. Moreover, in Section 5.8, a test dataset with the same geographic parameters but covering the period from January 2025-August 2025 will be used. More information regarding the test set will be presented in Section 5.8.4.

All waveforms and catalogues were accessed using `ObsPy` (Beyreuther et al., 2010). Every trace consists of around 200s of seismic displacement recorded by three orthogonal seismic channels (N,E,Z), which was interpolated into 4,096 evenly spaced samples, leading to a sampling rate of approximately 20 Hz. The instrument response was removed, and a bandpass filter from 0.1 to 8 Hz was applied to the waveforms (Van den Ende & Ampuero, 2020). Additionally, the waveform amplitudes were multiplied by a constant scaling factor of 1×10^8, as suggested by Dr Agius, since no specific scaling factor is established in the literature. Since recorded amplitudes are somewhat small, multiplying by a somewhat large factor raises the input data to a numerically stable range without losing magnitude information. A plot of the waveforms of the event with the largest magnitude can be seen in Figure 5.3.1.

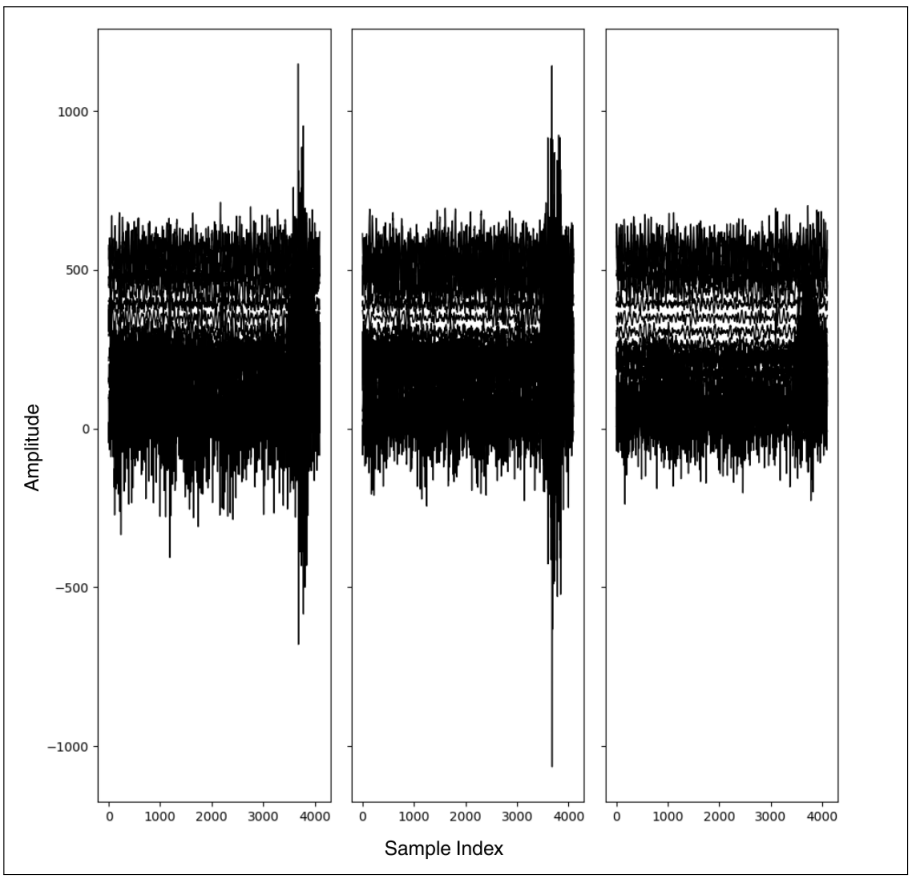

Figure 5.3.1: A plot of the waveforms of the event with the largest magnitude.

It is worth noting that seismic event recordings are sometimes interrupted due to issues such as power loss, resulting in incomplete waveform data. Consequently, all waveforms not having three channels were removed. In addition, certain stations may be non-functional during specific events because of outages, malfunctions, or temporary removal from the site. For these cases, the affected stations were omitted, as their waveform data lacked the quality and completeness required to serve as reliable input for NN training. Maps of events and stations for the data under study are illustrated in Figures 5.3.2 and 5.3.3, respectively.

Several statistical and spectral features were derived from the raw waveform metadata to provide the NN with additional predictor variables, thus enriching the input representation. These include the peak amplitude, root-mean square energy, spectral centroid and SNR, calculated independently for each of the three seismic components. These engineered features augment the original waveform data with physically meaningful characteristics, potentially leading to improve model performance. The Python code used to download the data is available in the GitHub repository, with the link provided in Appendix E. The subsequent section will now provide the model standards, training procedure and evaluation approaches, further supporting reproducibility and

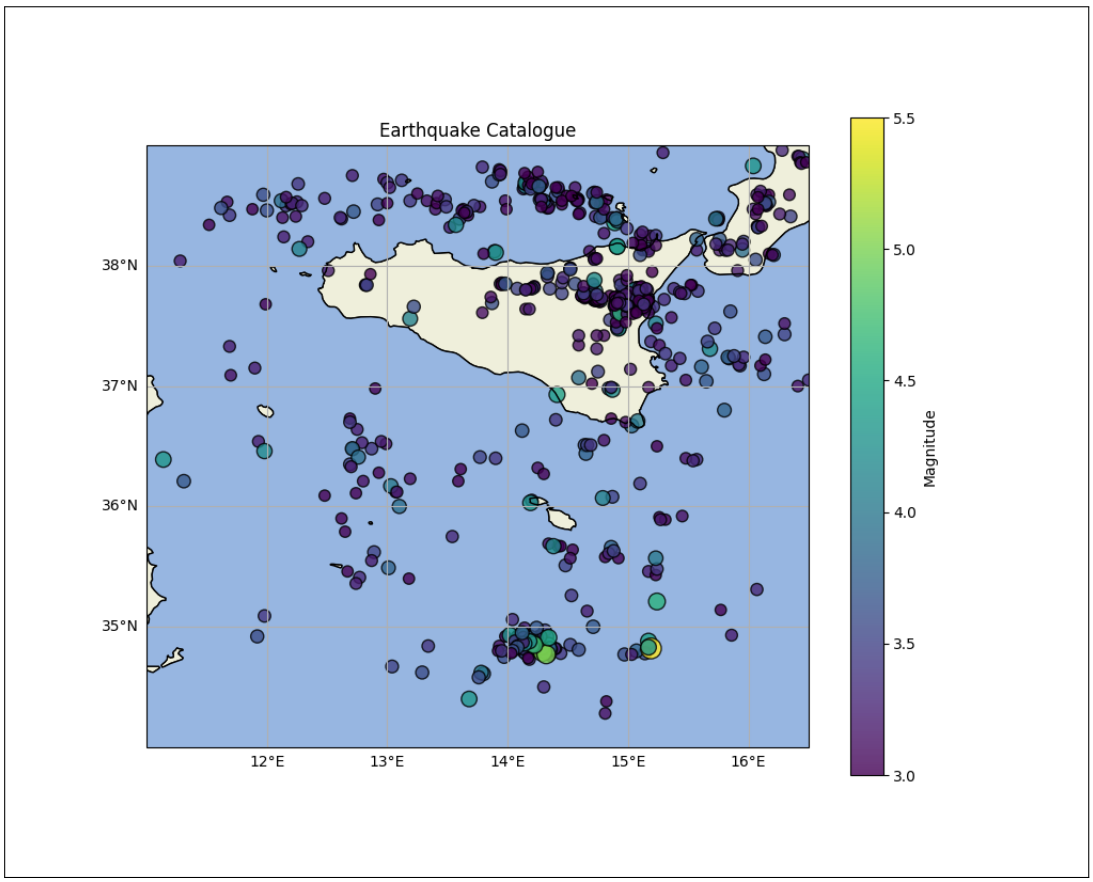

Figure 5.3.2: Map of the seismic events in the central Mediterranean region between January 2013 and December 2024.

comparative analysis.

5.4 Model Standards, Training Procedure and Model Evaluation

To ensure consistency across experiments and reproducibility of results, a global seed of 42 was set prior to all experimental runs. This is especially recommended when using operations that involve randomness, such as weight initialization of NN layers. All training processes were carried over 300 epochs, striking a suitable balance between allowing sufficient learning, while avoiding excessive computational burden. It is worth noting that, in accordance with Van den Ende & Ampuero (2020); X. Zhang et al. (2022), we initially attempted the experiments with 400–500 epochs, but this was not feasible on our hardware, as the local system exhausted RAM and the process was terminated. While proceeding with 300 epochs was feasible, this afforded the models less time to learn than in similar studies, which may be reflected in the prediction quality. Lastly, it is worth mentioning that attempts to implement parallel processing during hyperparameter tuning were unsuccessful, presumably due to memory constraints. Ac-

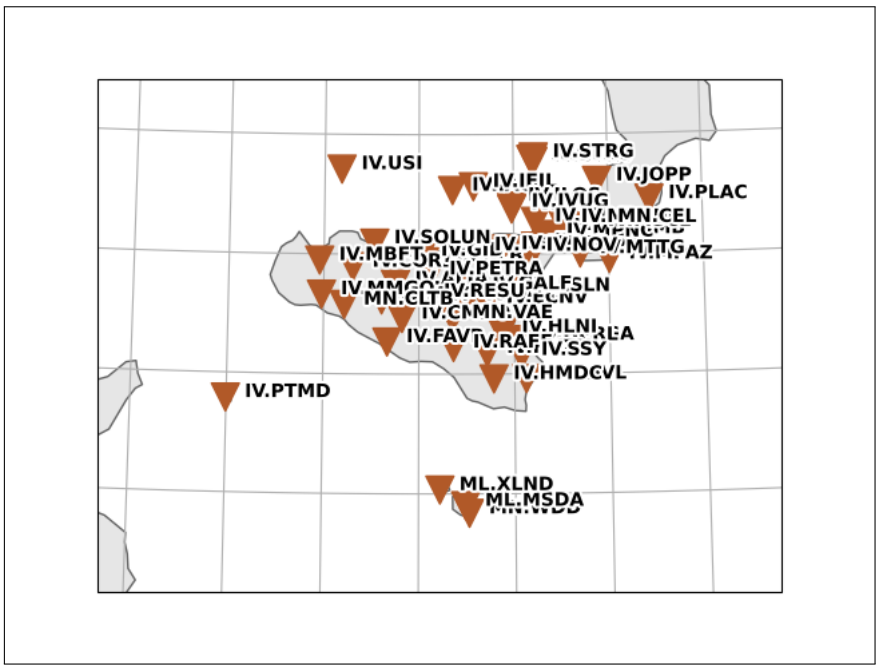

Figure 5.3.3: Map of the seismic stations recording information on the seismic events in the central Mediterranean region between January 2013 and December 2024.

cordingly, the tuning pipeline was executed serially, leading to an inevitable increase in computation time.

5.4.1 Regularization Techniques

To mitigate potential overfitting and improve generalization, we employed regularization throughout. In particular, spatial dropout and Gaussian dropout were applied across all models, with the dropout probability treated as a tunable hyperparameter. Similar studies have implemented the same regularization strategies (Van den Ende & Ampuero, 2020; X. Zhang et al., 2022), supporting our design choices. Moreover, for certain configurations we applied early stopping (patience = 20) monitoring validation loss.

5.4.2 Pre-Processing and Normalization

Following waveform pre-processing, the coordinates for both seismic stations and source locations are normalized using min-max scaler, so that they lie between -1 and 1. Analogously, the depth and magnitude are also normalized into the range [-1,1]. This step is a standard procedure in statistical learning tasks, as it facilitates faster convergence and model stability (X. Zhang et al., 2022).

The data splitting strategy consists of two stages to ensure independence between training and testing. All events from 2025 are held out as an independent test set for evaluation of predictive performance. The remaining dataset spanning 2013-

2024 is then randomly split into 80% training (548 events) and 20% validation (137 events), with the validation set used for hyperparameter tuning and model section. A fixed hold-out validation set is adopted due to computational constraints and relatively small dataset size when compared to similar studies (Van den Ende & Ampuero, 2020; X. Zhang et al., 2022). For each training iteration, batches are generated on-the-fly (i.e., happens during training instead of creating them and storing augmented copies beforehand), ensuring dynamic and memory efficient data loading. The batch size is a hyperparameter that will vary accordingly, as will be seen shortly in Section 5.5. Each event in a batch is represented by a fixed number of 40 stations, randomly chosen from the set of available stations for that event. In the case where an event is associated with less than 40 stations, the missing entries are zero padded. This is because the NN expects inputs to have a fixed tensor shape, otherwise the events cannot be batched together. So that the model performance is not affected, these zero inputs are then masked out during aggregation.

5.4.3 Loss Function and Optimizer

The model parameters are optimized via backpropagation by minimizing a modified version of the Huber loss function introduced in Section 2.4:

$$J_H(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{N_{train}} \sum_{k=1}^{N_{train}} \frac{1}{4} \sum_{j'=1}^{4} (\mathcal{J}_H(\hat{y}_{kj'}, y_{kj'})), \tag{5.1}$$

where

$$\mathcal{J}_H(\hat{y}_{kj'}, y_{kj'}) = \begin{cases} \frac{1}{2}(\hat{y}_{kj'} - y_{kj'})^2 & \text{if} \quad |\hat{y}_{kj'} - y_{kj'}| \leq \gamma \\ \gamma(|\hat{y}_{kj'} - y_{kj'}| - \frac{1}{2}\gamma) & \text{otherwise} \end{cases}.$$

where $\hat{\boldsymbol{y}} = (\hat{y}_{k1}, \hat{y}_{k2}, \hat{y}_{k3}, \hat{y}_{k4})$ denotes the model predictions for observation k, corresponding to the four source parameters *(latitude,longitude,depth,magnitude)* and $\boldsymbol{y} = (y_{k1}, y_{k2}, y_{k3}, y_{k4})$ represents the associated ground-truth values. Furthermore, N_{train} is the size of the training set and the inner summation over $j' = 1, ..., 4$ aggregates the loss across the four output variables for each observation. Lastly, $\mathcal{J}_H(\hat{y}_{kj'}, y_{kj'})$ is the Huber loss applied to the j'^{th} output of observation k and the parameter γ controls the transition point between the quadratic and the linear loss.

A value of 0.1 was arbitrarily selected for γ because it is small enough to preserve sensitivity to minor prediction errors via the quadratic component while also maintaining robustness against any potential outliers through the linear component. The loss is minimized using the AdamW optimizer with weight decay parameter, λ, equal

to $1e-4$ (X. Zhang et al., 2022). It is worth noting that the learning rate will be treated as a hyperparameter. Furthermore, the scaling parameter η_t is set to 1 for all iterations, so that the step size is controlled solely by the learning rate.

5.4.4 Gradient Computation and Backpropagation

As in the case of most NNs, training is performed through the backpropagation algorithm, which is automatically handled by the `TensorFlow` framework. This is achieved in the following way. During the forward pass, `TensorFlow` tracks the sequence of operations carried out. Then, `TensorFlow` traverses this list of operations in reverse order to compute gradients in the backward pass. In GNNs, the backpropagation algorithm is inherently constrained by the graph structure defined in the forward pass. During training, gradients are not propagated separately across all nodes, but instead follow the same neighbourhood aggregation rules encoded in the graph topology. This process is handled automatically by `TensorFlow`.

5.4.5 Evaluation of Performance

Firstly, in order to determine the best hyperparameter configurations, models are ranked according to validation MAE, MSE and NRMSE (see Section 2.7). Following this, training dynamics are assessed by plotting the training and validation Huber loss curves across epochs, in order to gain insight into convergence behaviour of the optimizer, as well as aid in identifying any signs of overfitting.

Additionally, scatter plots are generated to compare predicted versus actual values for the training and validation sets, with data points colour-coded (black for training and orange for validation). These plots allow us to assess both the model's ability to fit the training data and its capacity to generalize to unseen data. Data points lying close to the diagonal correspond to accurate predictions, whereas larger deviations indicate prediction errors comparing the spread and alignment of the training and validation points, therefore helps identify potential underfitting or overfitting.

In order to produced a predictive distribution, rather than single point estimates, an evaluation strategy based on Monte Carlo Dropout is employed by keeping Gaussian Dropout, (introduced in Section 3.5.2), active during inference for both training and validation sets. Retaining stochasticity at inference allows the model to generate multiple model predicted values for the same input. Each stochastic forward pass corresponds to a different effective subnetwork, enabling Monte Carlo Dropout to approximate model uncertainty through variability in the predicted values. For each

seismic event in the training and validation sets, the model performance 100 stochastic forward passes, yielding a distribution of predicted values for the four target variables: latitude, longitude, depth and magnitude. The mean of the predicted and standard deviation of these distributions are then calculated and unscaled to recover physical units, with the standard deviation utilized to construct the error bars shown in the scatter plots. Monte Carlo Dropout is not applied to the test set in order to preserve a single, unbiased estimate of final predictive performance. Instead, its use solely on training and validation sets supports uncertainty-aware fit and model evaluation through interpretation of "predictive" dispersion and comparison of model configuration.

After identifying the optimal models, defined here as those exhibiting stable training dynamics and strong generalisation performance on the validation set, their predictive ability is evaluated on an independent test set using the MAEP, MSEP and NRMSEP metrics (see Section 2.7), together with plots comparing predicted and actual values, to determine which model offers the best overall predictive performance.

5.5 Edgeless Graph Architecture

The first architecture considered will be referred to as the *edgeless graph architecture*. This is because the architecture processes input data structured as a graph where each node represents a seismic station, however, no edges are defined between the nodes. This design enables the architecture to learn from the individual attributes of each station and their contributions to the overall event characterization, without modelling explicit spatial or relational dependences between stations.

The architecture comprises several key components, specifically, an input layer, a CNN-based waveform encoder, a feature fusion block MLPs, a station weighting and aggregation mechanism, and a final MLP that produces the output vector of predictions containing the latitude, longitude, depth and magnitude predicted values of each seismic event. Section 5.5.1 provides a detailed breakdown of each component in the edgeless graph model architecture, inspired by the work of Van den Ende & Ampuero (2020). Any additional material pertaining to Section 5.5.1 can be found in Appendix F. The edgeless graph model architecture will serve as a baseline.

As will be seen in Section 5.5.2, in order to optimize model performance, a systematic hyperparameter tuning procedure will be performed using a grid search approach. Grid search was selected because it facilitates an exhaustive evaluation of predefined hyperparameter combinations, guaranteeing that the influence of each parameter on

model performance can be thoroughly evaluated. Lastly, in Section 5.5.3, early stopping will be added to the model with the intent of improving training performance.

5.5.1 Model Architecture

Input Layer

The input layer of the edgeless graph model is designed to intake information associated with individual seismic stations, that is, features attributed to the nodes of the graph. Four distinct input tensors, each capturing a different aspect of the station-level data, are fed into the input layer. The first input tensor, having shape (40, 4096, 3) contains the raw waveform metadata of 40 stations, where 4096 represents the number of time samples and 3 corresponds to the three waveform orthogonal seismic components, vertical (Z), North-South (N), and East-West (E). Additionally, a coordinate tensor of shape (40, 1, 3) provides the spatial location of each station, where the last dimension represents the latitude, longitude and elevation. A feature tensor of shape (40, 12) contains pre-computed, per-station signal characteristics, specifically, the peak amplitude, RMS amplitude, spectral centroid and SNR, extracted separately from each of the three seismic components. Lastly, a weight tensor of shape (40, 1) contains scalar weights for each station, allowing the model to adjust the contribution of individual nodes during the aggregation step.

CNN-Based Waveform Encoder

The CNN-based waveform encoder processes the seismic waveform input tensor of shape (batch size, 40, 4096, 3), where 40, 4096 and 3, once again refer to the number of stations, time samples and seismic components, respectively. In this context, the batch size dimension refers to the number of samples processed simultaneously during training and is typically required when applying batch normalization. Although the waveform is inherently a time series, 2D convolutions are used to leverage `TensorFlow`'s image-like operations and to convolve along the time dimension while preserving component separation. Specifically, a convolutional kernel of shape (1,5) is used to slide across time, treating the three components as channels, analogous to colour channels in an image. Typically, K_{N_t} (see Section 3.2.1.1) should be treated as a tunable hyperparameter, but because of the space and time constraint, a fixed value of 5 was selected, following the parameter setting used in Van den Ende & Ampuero (2020).

The encoder consists of three convolutional blocks, each containing three 2D convolutional layers. The number of filters doubles with each block, starting from a base of

2 filters and increasing as follows: 4→8→16. Each convolution is immediately followed by a ReLU activation function and Spatial Dropout, which help introduce nonlinear relationships and to reduce overfitting. The use of the ReLU activation function is motivated due to its ability to accelerate convergence and improve performance in deep architectures by mitigating vanishing gradients and encouraging sparse representations (Glorot & Bengio, 2010; Krizhevsky et al., 2012). Spatial dropout is employed because it is better suited for the mechanics of convolutional layers (Van den Ende & Ampuero, 2020). After each block, the time dimension is reduced using `MaxPooling2D` with pool size (1,4), effectively downsampling the temporal axis by a factor of 4 per block. Given an initial temporal length of 4096 samples, the downsampling proceeds as 4096→1024→256→64. Powers of two lengths were used for computational efficiency and to remain consistent with the Van den Ende & Ampuero (2020) architecture on which this architecture is based on. Thus, after the three blocks the output tensor has shape (batch size, 40, 64, 16).

Following these blocks, the encoder applies two additional 2D convolutional layers with kernel size (1,5) and 32 filters. The first of these layers is followed by a ReLU activation function and Spatial Dropout, while the second uses a tanh activation function without dropout. The use of tanh restricts the output values to the range $[-1, 1]$. No further downsampling is applied in these final layers, so the output shape becomes (batch size, 40, 64, 32). To summarize the temporal information for each station, a max reduction is applied across the time axis (axis=2), resulting in a tensor of shape (batch size, 40, 1, 32). This representation can be interpreted as a station-wise summary of the most salient temporal features. Finally, Gaussian Dropout is applied to this tensor using a specified dropout rate. Gaussian dropout was chosen over conventional dropout because it provides smoother, continuous noise injection (Srivastava et al., 2014). A schematic of the CNN-based waveform encoder can be found in Figure F.1.

Feature Fusion Block

Having processed the waveform data, the CNN output is concatenated with the station coordinates tensor having shape (batch size, 40, 1, 3) and the derived waveform features tensor having shape (batch size, 40, 1, 12), resulting in a final concatenated tensor of shape (batch size, 40, 1, 47). The tensor is then passed onto an MLP that processes the concatenated feature vector of each station independently along the spatial axis without affecting neighbouring stations. This is achieved through (1×1)-convolutional layers, where the convolutional kernels have a spatial size of (1×1). There are two

convolutional layers within this block. The first layer is a 2D-convolutional layer with 128 filters (activation maps) and a ReLU activation function followed by a spatial dropout layer for regularization. The number of filters is a convolutional layer is typically treated as a modifiable hyperparameter, however, due to space constraints, a fixed value of 128 filters was adopted, following the configuration used in Van den Ende & Ampuero (2020). The second layer is the same as the first, except that it does not include dropout. The shape of the tensor is maintained throughout this block as (batch size, 40, 1, 128). A schematic of the feature fusion block can be found in Figure F.2.

Station Weighting and Aggregation Mechanism

The station weighting and aggregation mechanism compresses the per-station features into a single global representation of the seismic event. In order to ensure that every station contributes appropriately, the model uses a set of learned scalar weights, which adjust the importance of each station during aggregation. Two inputs are considered, namely, the fused station features with shape (batch size, 40, 1, 128) and a weight tensor with shape (batch size, 40). The weight tensor is expanded to match the shape of the station features by adding a singleton dimension and repeating each shape 128 times, resulting in a shape of (batch size, 40, 1, 128).

Following this, each station's 128-dimensional feature vector is multiplied by its corresponding weight. This means that if a station has a lower weight, its features are scaled down, reducing its influence in the final aggregation and vice-versa. The result is a weighted version of the station's features, where each station's contribution reflects its reliability. Subsequently, information across stations is aggregated via global max pooling along the station axis, effectively choosing the most prominent feature values across all tensors. The resulting tensor, having shape (batch size, 1, 1, 128), is then flattened to produce a single 128-dimensional feature vector, which summarizes the entire event and is passed to the final prediction layers.

Output MLP

The final block of the model, responsible for obtaining the model's final output, is a fully connected MLP that maps the 128-dimensional feature vector into the four target variables, latitude, longitude, depth and magnitude. First, the input vector is passed through a fully connected layer with 128 units and a ReLU activation function, initialized using He initializer, because as discussed in Chapter 3, it caters for the ReLU activation function. Typically, the number of units is an adjustable hyperparameter,

however, it was chosen to retain 128 units to preserve the representational capacity learned in the preceding convolutional block. Then, Gaussian dropout, using a specified dropout probability, is applied to this layer's output to improve generalization to unseen data by introducing a level uncertainty into the model. This is followed by a second fully connected layer with 4 units and a tanh activation function. The result is a 4-dimensional output vector representing the model's predicted event latitude, longitude, depth and magnitude. A schematic of the output MLP used to generate the final predicted values can be seen in Figure F.3.

5.5.2 Hyperparameter Tuning

The hyperparameters considered in this experiment include batch size, learning rate and dropout probability. Batch sizes of 16, 32 and 64 were chosen so as to explore the trade-off between training stability and computational efficiency, as smaller batch sizes tend to improve generalization while larger batches can accelerate training. Learning rates of $1e-5, 1e-4, 1e-3$ and $1e-2$ were tested, capturing a wide range of possible convergence behaviours, ranging from very fine-grained updates to more aggressive optimization steps. Dropout probabilities of 0.2, 0.3, 0.4 and 0.5 were considered, as these are the most commonly used values in the NN literature (Srivastava et al., 2014). In total, 48 unique parameter configurations were evaluated. The whole hyperparameter tuning procedure took around 26.94 hours to complete. The Python code used for hyperparameter tuning is available in the GitHub repository, with the link provided in Appendix E.

Model performance was evaluated using the approach outlined in Section 5.4.5, with individual model configurations being chosen for each of the four target variables latitude, longitude, depth and magnitude, alongside the best-performing model overall. Due to space constraints, only the best five model configurations for each target variable, as well as, the top five overall models, are presented here. A comprehensive summary of all results can be found in Appendix G, Tables G.1-G.10.

Firstly, the optimal model configurations for each of the four target variables was determined separately. The best configuration will be chosen based on the lowest validation MAE, MSE and NRMSE values. In the cases where the metrics significantly disagree, the NRMSE will be given more importance due to it being a normalized and comparable measure of predictive accuracy across outputs. The top five model configurations for latitude differed slightly across validation metrics. The top five model configurations according to MAE can be found in Table 5.5.1, while those according

to MSE and NRMSE can be found in Table 5.5.2.

Batch Size	LR	Dropout Prob	MAE (°)
32	0.001	0.2	0.76
64	0.01	0.2	0.78
64	0.001	0.2	0.79
16	0.001	0.3	0.8
16	0.001	0.2	0.81

Table 5.5.1: Validation performance metrics of the best five hyperparameter combinations for latitude using the Edgeless Graph Architecture according to lowest MAE.

Batch Size	LR	Dropout Prob	MSE $(°)^2$	NRMSE
32	0.001	0.2	1.13	0.74
64	0.001	0.2	1.15	0.74
64	0.001	0.4	1.22	0.76
16	0.001	0.3	1.23	0.77
16	0.001	0.2	1.23	0.77

Table 5.5.2: Validation performance metrics of the best five hyperparameter combinations for latitude using the Edgeless Graph Architecture according to lowest MSE and NRMSE.

From Tables 5.5.1 and 5.5.2, the best model configuration for latitude, according to all three evaluation metrics, is batch size = 32, learning rate = 0.001 and dropout probability = 0.2. With regards to the target variable longitude, the top five model configurations ranked by MSE and NRMSE can be found in Table 5.5.3.

Batch Size	LR	Dropout Prob	MAE (°)	MSE $(°)^2$	NRMSE
16	0.001	0.3	0.65	0.79	0.99
64	0.01	0.2	0.65	0.79	0.99
32	0.001	0.4	0.65	0.79	0.99
16	0.001	0.5	0.65	0.79	0.99
32	0.001	0.2	0.64	0.8	1

Table 5.5.3: Validation performance metrics of the best five hyperparameter combinations for longitude using the Edgeless Graph Architecture according to lowest MSE and NRMSE.

Batch Size	LR	Dropout Prob	MAE (km)
16	0.001	0.5	6.19
16	0.001	0.4	6.22
32	0.001	0.5	6.25
32	0.00001	0.3	6.25
32	0.001	0.4	6.26

Table 5.5.4: Validation performance metrics of the best five hyperparameter combinations for depth using the Edgeless Graph Architecture according to lowest MAE.

Upon inspecting Tables 5.5.4 and 5.5.5, it was decided to choose the configuration with batch size = 32, learning rate = 0.00001, and dropout probability = 0.3 as the optimal hyperparameter combination for depth. This is because it achieved the lowest MSE and NRMSE values, while maintaining an MAE comparable to the best MAE

Batch Size	LR	Dropout Prob	MSE $(km)^2$	NRMSE
32	0.00001	0.3	52.79	0.992
32	0.01	0.3	53.31	0.997
64	0.01	0.4	53.31	0.997
16	0.001	0.5	53.48	1
32	0.0001	0.3	53.51	1

Table 5.5.5: Validation performance metrics of the best five hyperparameter combinations for depth using the Edgeless Graph Architecture according to lowest MSE and NRMSE.

value. This balance makes it the most reliable choice among the tested configuration. The top five model configurations for the magnitude target variable can be found in Table 5.5.6.

Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
32	0.001	0.2	0.27	0.11	0.94
16	0.001	0.5	0.27	0.12	0.98
32	0.001	0.4	0.27	0.12	0.98
16	0.001	0.3	0.27	0.12	0.98
32	0.001	0.3	0.27	0.12	0.98

Table 5.5.6: Validation performance metrics of the best five hyperparameter combinations for magnitude using the Edgeless Graph Architecture (ordered by lowest NRMSE).

From Table 5.5.6 it can be concluded that according to all three metrics, the best hyperparameter configuration is batch size = 32, learning rate = 0.001, and dropout probability = 0.2. Lastly, since the target variables are measured in different units, NRMSE was used to determine the best overall model, as it provides a unitless and comparable performance metric across the four target variables. The overall NRMSE is obtained by averaging the NRMSE values across all target variables. The overall NRMSE values for the top five model configurations are found in Table 5.5.7. From Table 5.5.7, it can be seen that the overall best model corresponds to the configuration batch size = 32, learning rate = 0.001, and dropout probability = 0.2.

Batch Size	LR	Dropout Prob	NRMSE
32	0.001	0.2	0.931
16	0.001	0.3	0.939
64	0.001	0.4	0.942
64	0.001	0.2	0.944
64	0.01	0.2	0.945

Table 5.5.7: Overall validation NRMSE values of the best five hyperparameter combinations for the four target variables using the Edgeless Graph Architecture.

A summary of the optimal hyperparameter configurations for each target variable and the overall best model configuration based on NRMSE using Edgeless Graph Architecture is presented in Table 5.5.8. Table 5.5.8 shows that the best model con-

figuration for almost all the target variables, as well as the overall best model based on NRMSE using the edgeless graph architecture is the same, having batch size = 32, learning rate = 0.001 and dropout probability = 0.2. This will be referred to as Model 1 from now onwards. For the depth variable, the best configuration is the one having batch size = 32, learning rate = 0.00001 and dropout probability = 0.3. This will be referred to as Model 2. Plots of training and validation loss per epoch for Model 1 and Model 2 are presented in Figure 5.5.1 and 5.5.2.

Variable	**Batch Size**	**LR**	**Dropout Prob**	**MAE**	**MSE**	**NRMSE**
Latitude	32	0.001	0.2	0.76	1.13	0.74
Longitude	32	0.001	0.2	0.65	0.79	0.99
Depth	32	0.00001	0.3	6.19	52.79	0.992
Magnitude	32	0.001	0.2	0.27	0.11	0.937
All	32	0.001	0.2			0.931

Table 5.5.8: Summary of the optimal hyperparameter configurations for each target variable and the overall best hyperparameter configuration based on validation NRMSE using Edgeless Graph Architecture.

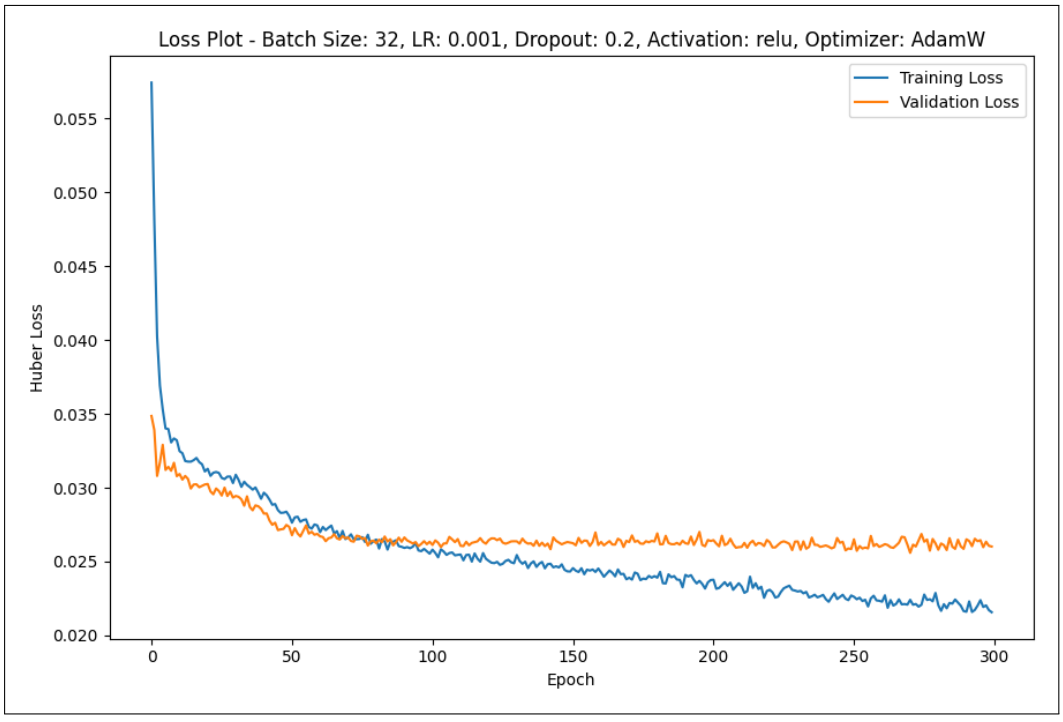

Figure 5.5.1: Loss Plot for Model 1.

The loss curve demonstrates how the model's error diminishes while it learns and thus, allows us to assess training dynamics. When loss curves are interpreted, three properties are usually analysed, namely, *smoothness*, *convergence* and *generalization*. An ideal loss plot should be smooth as it is indicative of gradual and consistent changes in the model's performance as it goes through training. When the loss curve is smooth, it implies that the model is learning at a stable and consistent rate. A loss curve is

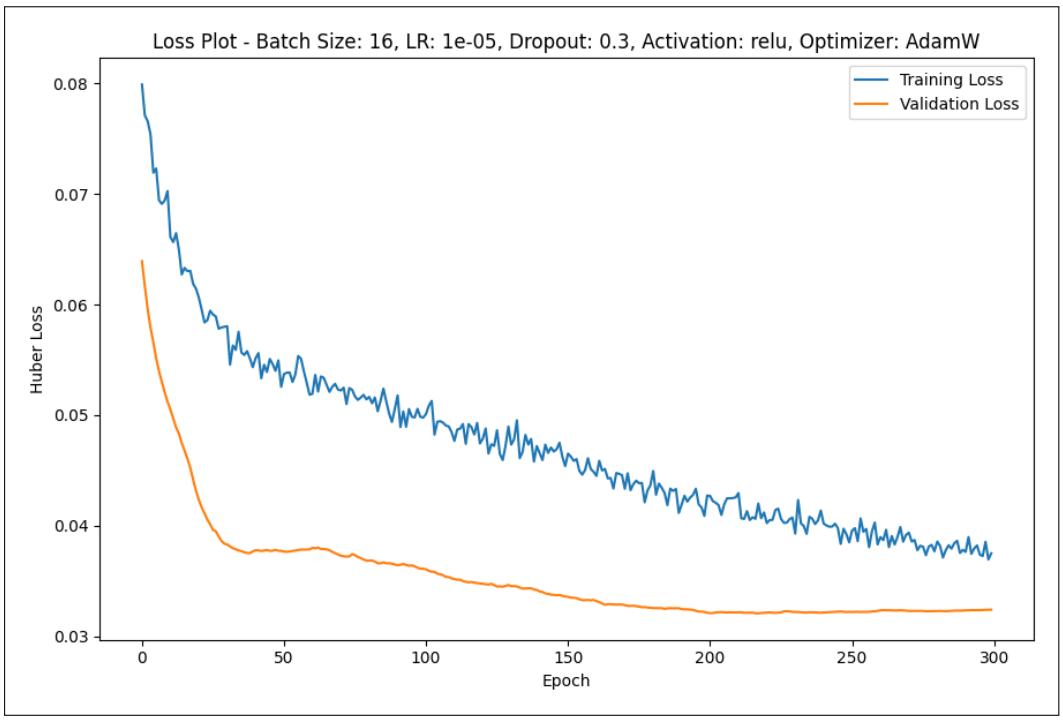

Figure 5.5.2: Loss Plot for Model 2.

said to converge when it plateaus. This means that a loss curve would have reached a stable or optimal rate. When a loss curve converges to a point, it implies that further training will unlikely lead to any significant improvements in model fit. This means that the model has learned all it could from the training data and cannot improve further. From the loss curve, one can determine whether a model has the ability to generalize well, i.e., rather than performing well on training data alone, it can also perform well on new and unseen data. This occurs when the difference between the model's training and validation loss is small.

In terms of smoothness, the loss curves in Figure 5.5.1 exhibit mild oscillations, reflecting normal stochastic variation due to relatively larger parameter updates near convergence rather than training instability. In contrast, the loss curves in Figure 5.5.2 appear smoother, particularly for the validation loss; however, this behaviour is primarily attributable to the much smaller learning rate, which results in smaller and more gradual parameter updates, rather than improved optimisation dynamics. Furthermore, the loss curves for Model 1 plateau, while those for Model 2 do not. This suggests Model 1 converges better than Model 2. It is worth noting that the loss curves for Model 1 seem to plateau around the 100^{th} epoch with very little improvement afterwards. This suggests that extending training beyond 150-200 epochs yields diminishing returns, and the model could benefit from early stopping. Lastly, the training-validation gap for Model 1 is small and stable with no sharp deviations,

whereas for Model 2 the validation loss drops below the training loss, an unusual behaviour that suggests underfitting and weaker generalization.

Figures 5.5.3 and 5.5.4 illustrate the predicted versus target values for Model 1, while Figures 5.5.5 and 5.5.6 illustrate the predicted versus target values for Model 2. Training results depicted in black and validation results in orange. The dashed diagonal line indicates the ideal scenario where predictions match the actual values exactly. The closeness of the points to this line provides an intuitive evaluation of model fit and generalization ability as well as, accuracy, while the corresponding error bars highlight uncertainty in prediction. Training-validation metrics will also be examined as they provide insights into model generalisation.

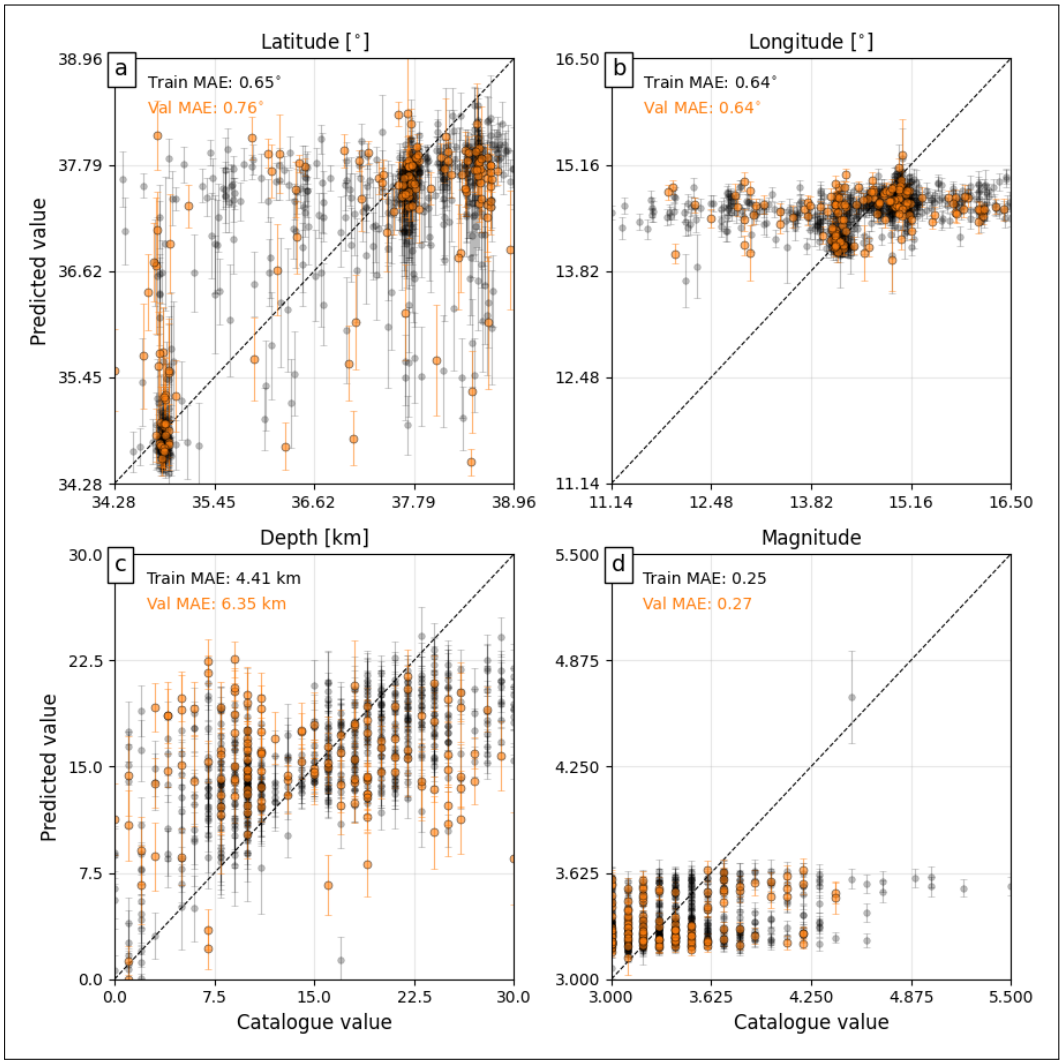

Figure 5.5.3: Actual versus predicted target values for Model 1 with corresponding MAE computed on training and validation sets.

Comparing the two models, the plots in Figures 5.5.3 and 5.5.4 seem to show training and validation predicted values that align more closely with the diagonal, indicating that Model 1 is better at tracking the true values across all target variables.

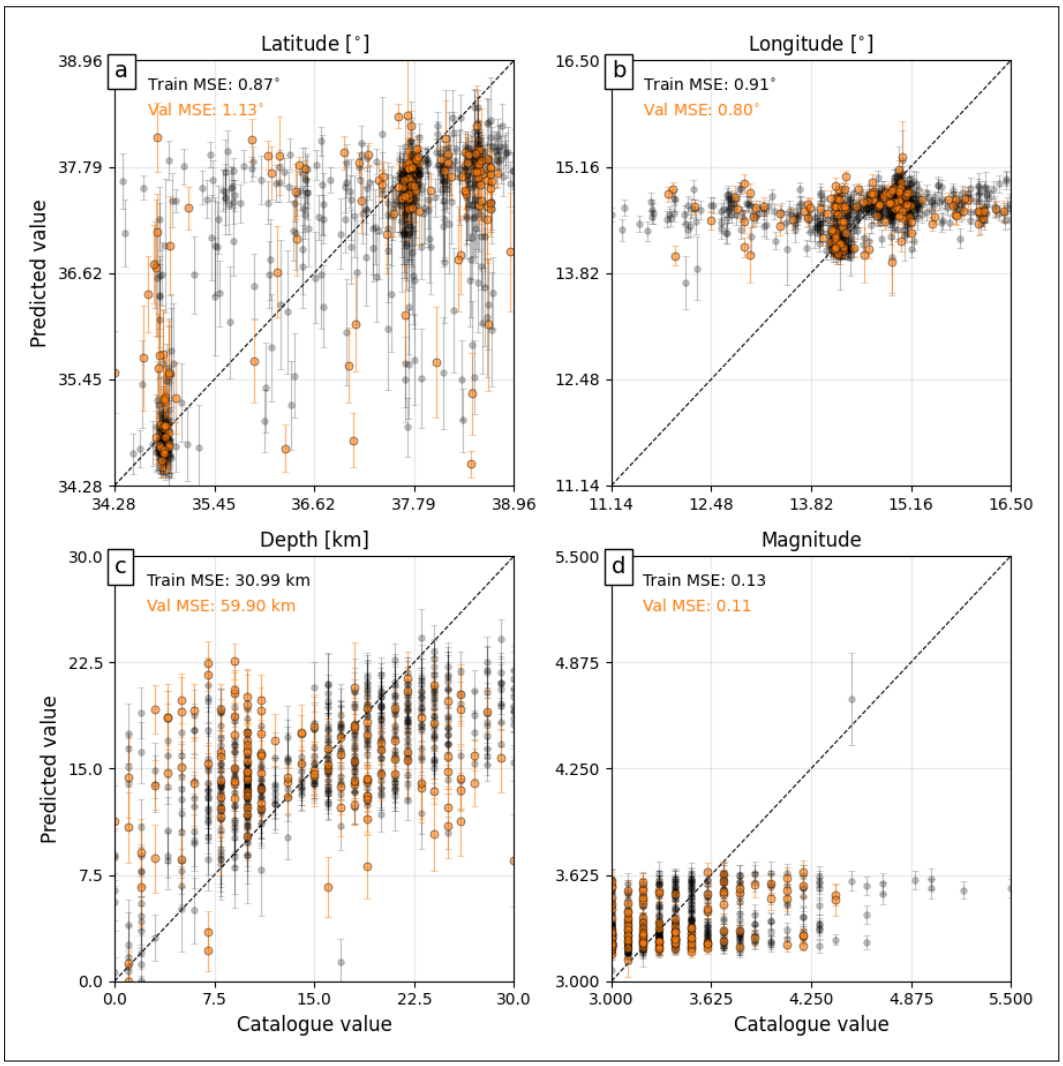

Figure 5.5.4: Actual versus predicted target values for for Model 1 with corresponding MSE computed on training and validation sets.

This means that Model 1 potentially fits the data better than Model 2. Additionally, Model 1 might generalize better. For the latitude variable, the predicted values for Model 1 form somewhat of a sloped trend, with lower and higher catalogue latitudes being predicted considerably well since many points follow the diagonal line. Conversely, the latitude predicted values for Model 2 form a horizontal band around 37.8°. Additionally, Model 1 exhibits smaller error bars and MAE/MSE values when compared to Model 2. With regards to the longitude target variable, both models' predicted values form a horizontal band at around 16.1°, indicating underfitting. Both models seem to largely ignore the variation in the catalogue longitude values and instead predict close to the mean. In addition to this, the error bars become wider away from the mean, indicating greater uncertainty for longitude further away from the mean value. It is worth noting however that Model 1 exhibits significantly smaller errors and error bars when compared to Model 2.

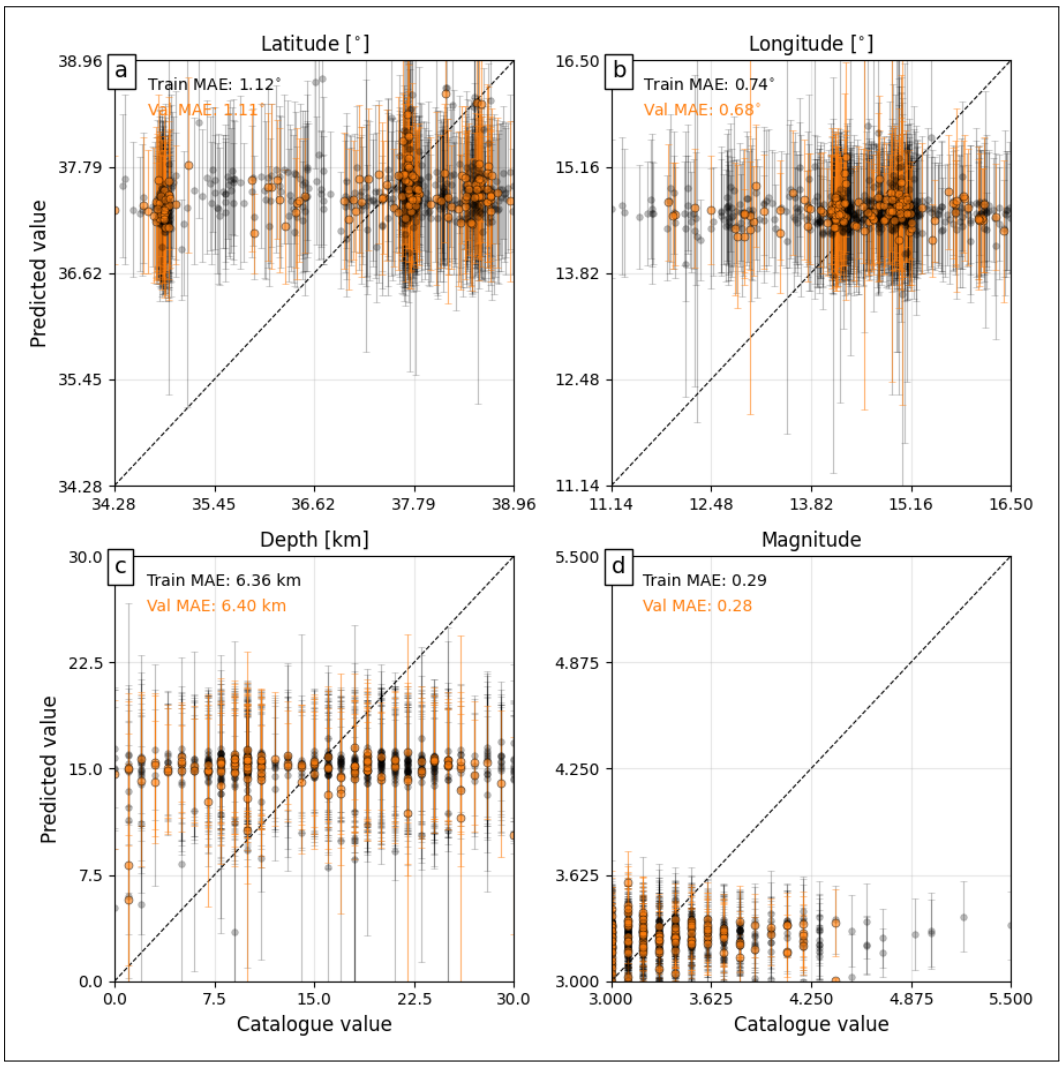

Figure 5.5.5: Actual versus predicted target values for Model 2 with corresponding MAE computed on training and validation sets.

For the depth target variable, Model 1's predicted values seem to follow the diagonal line more tightly with smaller errors and error bars when compared to Model 2. However, when compared to the results for the other target variables, Model 1's depth predictions still show the strongest evidence of underfitting with predictions concentrated around 8-18km, regardless of true depth. This results in consistent overestimation of shallow earthquakes and underestimation of deeper earthquakes. This is further corroborated by the larger error bars at shallow/greater depths. The metrics further highlight this weakness since training errors are already quite high (MAE=4.41 and MSE=30.99km), and validation errors increase significantly (MAE=6.35 and MSE=59.9km).

With regards to magnitude, comparing Figures 5.5.3 and 5.5.4 with 5.5.5 and 5.5.6, it can be seen that Model 2 underestimates magnitude when compared with Model 1. Additionally, Model 2's errors and error bars are significantly larger. Having said that, Model 1's magnitude predictions still form a horizontal band between 3.2 and

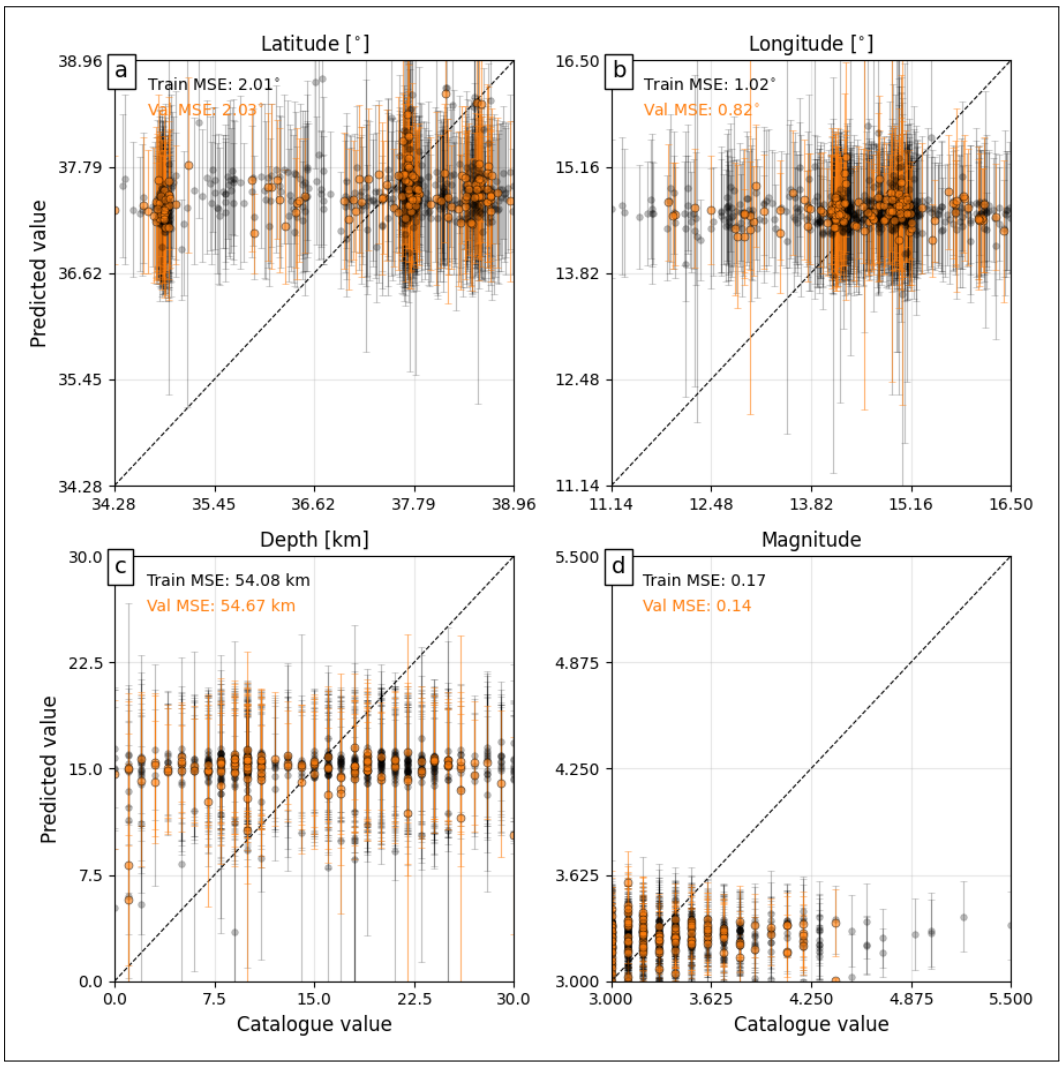

Figure 5.5.6: Actual versus predicted target values for for Model 2 with corresponding MSE computed on training and validation sets.

3.6, showing some underfitting with consistent under estimation of higher magnitude values. Training and validation results are almost identical (train MAE=0.25 and validation MAE=0.27; train MSE=0.11 and validation MSE=0.13), showing no signs of overfitting. However, it is worth mentioning that the low and comparable error might be attributed to the narrow magnitude range. Finally, the error bars are small, but still slightly expand at the higher end of the magnitude scale, demonstrating increasing uncertainty for stronger events.

Taking everything into consideration, from the results of the loss plots and actual versus predicted target values plots, Model 1 outperforms Model 2 in model fit, generalization ability and accuracy. This is because it exhibited better training dynamics, showed better generalization and had less model uncertainty. Thus, Model 1 will be selected as the best edgeless graph model so far. Table 5.5.9 shows a summary of the validation performance metrics for Model 1. It is worth noting that there is still some

underfitting present. The worst results were for the depth target variable, while the best results were for the magnitude target variable. This is consistent with findings in the literature (Van den Ende & Ampuero, 2020; X. Zhang et al., 2022).

Variable	MAE	MSE	NRMSE
Latitude	0.76	1.13	0.74
Longitude	0.65	0.79	0.99
Depth	6.35	59.9	1.06
Magnitude	0.27	0.11	0.937
All			0.931

Table 5.5.9: Summary of Validation Performance Metrics for Model 1: batch size=32, learning rate=0.001, dropout probability=0.2.

The underfitting and model uncertainty that resulted for Model 1 could be attributed to the fact that the events and stations in the dataset are not evenly distributed as seen in Figures 5.3.2 and 5.3.3. The use of Monte Carlo dropout could have also worsened the model uncertainty since its variational approximation is crude and often leads to underestimated predictive variance (Gal & Ghahramani, 2016; Ovadia et al., 2019). Nevertheless, Monte Carlo dropout was used to estimate uncertainty because it provides computationally inexpensive approximation to Bayesian inference NNs (Gal & Ghahramani, 2016) and because it was used in a study similar to the one being conducted in this dissertation (Van den Ende & Ampuero, 2020). The loss plot for Model 1 in Figure 5.5.1 suggests that Model 1 might potentially benefit from some early stopping.

5.5.3 Early Stopping

As discussed in Section 5.5.2, it was decided to refit Model 1 on the data, keeping all parameters the same, however, also adding early stopping with `patience` = 20 epochs. This model will be called Model 3. A summary of the resulting validation metrics is presented in Table 5.5.10. Due to space constraints the remaining loss plots and actual versus predicted target values plots for the edgeless graph models are presented in Appendix H. Figure H.1 shows the loss plot for Model 3, while Figures H.7 and H.8 illustrate the predicted versus target values. Notably, training stopped after 256 epochs, with the best weights restored from epoch 236, as determined by the early stopping criterion.

Comparing the metrics from Table 5.5.10 with those in Table 5.5.8, it can be seen that validation metrics are comparable between models. This suggests that adding early stopping does not worsen or improve model performance. Comparing the loss plots in Figures 5.5.1 and H.1, it can be seen that adding early stopping to our model

Variable	MAE	MSE	NRMSE
Latitude	0.78	1.19	0.76
Longitude	0.64	0.79	0.99
Depth	6.31	55.03	1.01
Magnitude	0.27	0.12	0.98
All			0.94

Table 5.5.10: Summary of Validation Performance Metrics for Model 3: batch size=32, learning rate=0.001, dropout probability=0.2 with early stopping patience=20.

improves training dynamics as the training and validation losses decrease smoothly together, showing stable convergence, minimal overfitting and better generalization. Lastly, upon examining Figures H.7 and H.8, it can be observed that Model 3's predictions are comparable to Model 1's. Even though the training and validation errors are slightly larger, the gap between training-validation errors is smaller. Taking everything into consideration it can be concluded that adding early stopping to our model leaves accuracy unchanged but improves training dynamics and reduces training time. As a result, it was decided to retain Model 3 as the best edgeless graph model.

5.6 GNN Architecture with Dynamic Edges

The second architecture considered will be referred to as the *dynamic edges GNN architecture*. The name is attributed to the novel innovation introduced to the model architecture. Specifically, edges are dynamically generated between the stations based on both geographic proximity and feature similarity, enabling graph based message passing, as discussed in Section 4.3.1. The dynamic edges construction procedure was inspired by the work of X. Zhang et al. (2022). As will be seen in Section 5.6.1, the dynamic edges GNN architecture maintains a similar architecture to that of the edgeless graph architecture discussed in Section 5.5, consisting of an input layer, a CNN-based waveform encoder, a feature fusion block, a station weighting and aggregation mechanism, and a final MLP that produces the output predictions. As before, any additional material pertaining to Section 5.6.1 can be found in Appendix F. Hyperparameter tuning and early stopping are also conducted for the dynamic edges GNN models, with results presented in Section 5.6.2, respectively. Once again due to space constraints, the loss plots and actual versus predicted target values plots pertaining to the dynamic edges GNN models are presented in Appendix J.

5.6.1 Model Architecture

The only architectural difference from the edgeless graph model architecture lies in the feature fusion block. Instead of a simple shared MLP, the fused per-station features are

passed into a graph-based fusion module composed of four stacked `FeatureFusionLayer` block. A schematic of this feature fusion block can be found in Figure F.4. The number of feature fusion blocks can be treated as a hyperparameter, however, due to computational constraints, it was decided to follow the parameter setting utilized by X. Zhang et al. (2022). Each of these blocks performs message passing over two separate k-nearest neighbour-graphs, one constructed from the geographic distance between stations, and the other from feature similarity. For each graph, the model computes differences between a station and its neighbours, processes those differences using learnable transformations, and then applies max aggregation across neighbours to update the station's feature vector. The detailed outline of the theoretical underpinnings of this process can be found in Section 4.4. Then, the outputs from all four graph layers are concatenated along the feature dimension, yielding a representation of shape (batch size, 40, 128). This is then reshaped to (batch size, 40, 1, 128) to match the expected input shape of the next block. The Python code used containing the dynamic edges GNN model architecture is available in the GitHub repository, with the link provided in Appendix E.

5.6.2 Hyperparameter Tuning

Analogous to the approach described in Section 5.5.2, hyperparameter tuning was performed to identify the optimal hyperparameter configurations. The same hyperparameter values for the batch size, learning rate and dropout probability were retained, with the addition of a new hyperparameter, k, which represents the maximum number of neighbours in the graph. For this experiment, k, was set to values 3 and 4. Larger values for k could not be considered, because the local machine ran out of RAM and terminated the process. There were a total of 96 model configurations and the entire hyperparameter tuning procedure took approximately 59.15 hours to complete.

Model performance was once again evaluated using the strategy outlined in Section 5.4.5, with separate model configurations being selected for each of the four target variables latitude, longitude, depth and magnitude, alongside the overall best-performing model. As before, due to space constraints, only the best five model configurations for each target variable, as well as, the top five overall models, are presented here. A comprehensive summary of all results can be found in Appendix I Tables I.1-I.15.

Firstly, the optimal model configuration for each of the four target variables was determined individually. The validation performance metrics for the top five model

configurations corresponding to latitude can be found in Table 5.6.1.

k	**Batch Size**	**LR**	**Dropout Prob**	**MAE (°)**	**MSE (°)^2**	**NRMSE**
4	32	0.001	0.4	0.8	1.34	0.80
4	64	0.001	0.3	0.82	1.4	0.82
4	16	0.001	0.4	0.82	1.41	0.82
3	64	0.01	0.2	0.79	1.44	0.83
3	32	0.001	0.4	0.88	1.48	0.84

Table 5.6.1: Validation performance metrics of the best five hyperparameter combinations for latitude using the Dynamic Edges GNN Architecture according to lowest metrics.

From 5.6.1 it can be seen that, according to MSE and NRMSE the best hyperparameter configuration for the latitude target variable is $k = 4$, batch size $= 32$, learning rate $= 0.001$ and dropout probability $= 0.4$. This is also the second best hyperparameter combination according to MAE, and so will be selected as the best hyperparameter configuration for the latitude target variable. The top five models for longitude differ according to validation performance metrics. Table 5.6.2 shows the top five longitude models according to MAE, while Table 5.6.3 shows the top five longitude models according to MSE and NRMSE.

k	**Batch Size**	**LR**	**Dropout Prob**	**MAE (°)**
3	32	0.001	0.4	0.62
4	16	0.001	0.2	0.63
3	16	0.001	0.2	0.63
4	32	0.001	0.4	0.64
3	64	0.01	0.2	0.64

Table 5.6.2: Validation performance metrics of the best five hyperparameter combinations for longitude using the Dynamic Edges GNN Architecture according to lowest MAE.

k	**Batch Size**	**LR**	**Dropout Prob**	**MSE (°)^2**	**NRMSE**
3	32	0.001	0.4	0.79	0.990
3	16	0.001	0.5	0.79	0.990
4	16	0.001	0.2	0.8	0.996
4	64	0.01	0.4	0.8	0.996
3	32	0.01	0.2	0.8	0.996

Table 5.6.3: Validation performance metrics of the best five hyperparameter combinations for longitude using the Dynamic Edges GNN Architecture according to lowest MSE and NRMSE.

From Tables 5.6.2 and 5.6.3, it can be seen that the best performing model configuration for longitude is $k = 3$, batch size $= 32$, learning rate $= 0.001$ and dropout probability $= 0.4$. The top five model configurations for depth differ according to performance metrics. Table 5.6.4 shows the top five depth models according to MAE, while Table 5.6.5 shows the top five depth models according to MSE and NRMSE.

k	**Batch Size**	**LR**	**Dropout Prob**	**MAE (km)**
4	32	0.0001	0.5	6.15
4	32	0.01	0.3	6.21
4	32	0.001	0.4	6.22
4	32	0.001	0.5	6.23
4	16	0.0001	0.4	6.26

Table 5.6.4: Validation performance metrics of the best five hyperparameter combinations for depth using the Dynamic Edges GNN Architecture according to lowest MAE.

k	**Batch Size**	**LR**	**Dropout Prob**	**MSE (km)^2**	**NRMSE**
4	32	0.01	0.3	51.93	0.984
4	32	0.001	0.5	52.46	0.989
4	32	0.0001	0.2	53.07	0.995
3	64	0.0001	0.2	53.44	0.998
3	64	0.001	0.4	53.78	1

Table 5.6.5: Validation performance metrics of the best five hyperparameter combinations for depth using the Dynamic Edges GNN Architecture according to lowest MSE and NRMSE.

With regards to the depth target variable, the top five model configurations vary considerably across metrics as can be seen from Tables 5.6.4 and 5.6.5. However, it was decided to choose the parameter configuration $k = 4$, batch size = 32, learning rate = 0.01 and dropout probability = 0.3, as it achieved the lowest MSE and NRMSE values, and the second lowest MAE value. The top five model configurations for the magnitude variable are presented in Table 5.6.6.

k	**Batch Size**	**LR**	**Dropout Prob**	**MAE**	**MSE**	**NRMSE**
3	32	0.001	0.4	0.12	0.98	0.28
3	32	0.001	0.3	0.12	0.98	0.27
3	16	0.001	0.3	0.12	0.98	0.27
3	64	0.001	0.2	0.12	0.98	0.27
3	16	0.001	0.2	0.12	0.98	0.27

Table 5.6.6: Validation performance metrics of the best five hyperparameter combinations for magnitude using the Dynamic Edges GNN Architecture (ordered by lowest MAE).

From Table 5.6.6, although the configuration $k = 3$, batch size = 32, learning rate = 0.001 and dropout probability = 0.4 yields the second-best MAE, the difference from the best is negligible. Moreover, MSE and NRMSE are comparable across all five configurations. Therefore this parameter setting was chosen to maintain consistency with the best longitude model configuration, which uses the same hyperparameters. Lastly, since the target variables are measured in different units, analogous to the edgeless graph model, NRMSE is used to identify the best overall dynamic edges GNN model, as it provides a unitless and comparable performance metric across all target variables. The overall NRMSE values for the top five model configurations are reported in Table 5.6.7. From Table 5.6.7, it can be seen that the overall best model

corresponds to the configuration $k = 3$, batch size = 32, learning rate = 0.001 and dropout probability = 0.4.

k	**Batch Size**	**LR**	**Dropout Prob**	**NRMSE**
3	32	0.001	0.4	0.95954
4	32	0.001	0.4	0.95956
4	64	0.01	0.2	0.9635
4	64	0.001	0.3	0.96512
4	64	0.01	0.4	0.9689

Table 5.6.7: Overall validation NRMSE values of the best five hyperparameter combinations for the four target variables using the Dynamic Edges GNN Architecture.

A summary of the optimal hyperparameter configurations for each target variable and the overall best model based on NRMSE using dynamic edges GNN model is presented in Table 5.6.8. Table 5.6.8 shows that for the target variables longitude and magnitude, the best parameter configuration is setting $k = 3$, batch size = 32, learning rate = 0.001 and dropout probability = 0.4. These were also the parameter settings achieved for the overall best model based on NRMSE using the dynamic edges GNN architecture. This model will be referred to as Model 4. For the latitude target variable, the optimal parameter configuration is setting $k = 4$, batch size = 32, learning rate = 0.001 and dropout probability = 0.4. This model will be referred to as Model 5. Lastly, for depth the optimal parameter configuration is $k = 4$, batch size = 32, learning rate = 0.01 and dropout probability = 0.3. This model will be referred to as Model 6.

Now, loss plots and actual versus predicted target values plots will be examined to assess training dynamics, generalization ability and accuracy. The loss plots for Models 4-6 are found in Figures J.1-J.3, while the predictions plots can be found in Figures J.8-J.14.

Variable	k	**Batch Size**	**LR**	**Dropout Prob**	**MAE**	**MSE**	**NRMSE**
Latitude	4	32	0.001	0.4	0.8	1.34	0.8
Longitude	3	32	0.001	0.4	0.62	0.79	0.99
Depth	4	32	0.01	0.3	6.21	51.93	0.984
Magnitude	3	32	0.001	0.4	0.28	0.12	0.98
All	3	32	0.001	0.4			0.96

Table 5.6.8: Summary of the optimal hyperparameter configurations for each target variable and the overall best hyperparameter configuration based on validation NRMSE using Dynamic Edges GNN Architecture.

Upon inspecting the loss plots in Figures J.1-J.3, Model 6 seems to be the best performing model in terms of training dynamics. This is because it has the smoothest loss curves, the fastest convergence, and a nearly perfect overlap between training and

validation losses, whereas Models 4 and 5 converge more slowly and exhibit plateaus before sharper drops later in training. However, as can be observed from Figures J.13 and J.14, Model 6 suffers from severe underfitting, with predictions collapsing into tight horizontal bands across all variables and producing the largest errors overall, particularly for depth and latitude.

In contrast, Models 4 and 5 both demonstrate reasonable alignment with the diagonal and comparable levels of scatter, with Model 5 showing slightly tighter clustering along the diagonal line for latitude and depth, but Model 4 achieving stronger validation metrics in two of the four target variables and lower overall errors. Between these two models, the differences are small, however, Model 4 achieved the best validation performance in two of the four target variables and the best overall validation NRMSE (see Table 5.6.8). Therefore, Model 4 is chosen as the best dynamic GNN edges model. A table with the validation metrics for each target variable for Model 4 is presented in Table 5.6.9.

Variable	**MAE**	**MSE**	**NRMSE**
Latitude	0.88	1.48	0.84
Longitude	0.62	0.79	0.990
Depth	6.44	56.63	1.027
Magnitude	0.28	0.12	0.98
All			0.96

Table 5.6.9: Summary of Validation Performance Metrics for Model 4: $k = 3$, batch size=32, learning rate=0.001, dropout probability=0.4.

5.7 Cluster Analysis

Seismic activity in the central Mediterranean region is rarely evenly distributed, as events and stations are often concentrated in localized pockets where geological conditions favour faulting. As illustrated in Figures 5.3.2 and 5.3.3, the dense collections of epicentres and nearby stations suggest the existence of hidden subregions with distinct propagation and noise characteristics. In the absence of any prior labelling knowledge, by applying cluster analysis to event locations, we can partition the study area into potentially homogeneous zones. It is hoped that this unsupervised approach reveals natural groupings of earthquakes and will aid in tailoring the predictive models to smaller, more focused regions, potentially improving localization accuracy and computational efficiency. All relevant theory pertaining to clustering can be found in Appendix K, while the Python code used to carry out the analysis is available in the GitHub repository, with the link provided in Appendix E.

Prior to performing cluster analysis, a seed of 42 was set to ensure repeatability and consistency across results. Then, the data was standardized using the `StandardScaler()` function in Python. Following this, the Hopkins' statistic (Hopkins & Skellam, 1954) was calculated to determine whether there is in fact a clustering structure present within the data. The Hopkins' statistic was chosen as it is typically the preferred method in the literature (Han et al., 2012). A detailed explanation of the Hopkins' statistic can be found in Appendix K, Section K.4. The Hopkins' statistic ranges from 0 to 1 with values above 0.75 typically indicative of a strong clustering tendency (Lawson & Jurs, 1990). In our study, the Hopkins' statistic was found to be approximately 0.85, which is relatively high and indicates strong evidence of a clustering structure. Thus, it was decided to apply clustering to the seismic data.

Three clustering methods were considered in this application, namely, K-means (MacQueen, 1967; Steinhaus, 1957), DBSCAN (Ester et al., 1996), and HDBSCAN (Campello et al., 2013). K-means was included because it is one of the simplest and most widely used techniques in the cluster analysis literature. DBSCAN was selected next as a density-based method well suited for discovering clusters of arbitrary (including non-convex) shape and for handling any potential outliers in the data. Finally, HDBSCAN was adopted to overcome DBSCAN's limitation of using a single, global density threshold. HDBSCAN achieves this by constructing a hierarchy of clusters across varying density levels, identifying nested structures and clusters of differing densities within the same dataset. More information on K-means, DBSCAN and HDBSCAN can be found in Appendix K Sections K.1, K.2 and K.3, respectively.

The first clustering technique to be considered was K-means. This method requires the pre-specification of the optimal number of clusters. There are numerous methods within the literature to determine the optimal number of clusters. In this application, the elbow method (Thorndike, 1953), silhouette score (Rousseeuw, 1987) and gap statistic (Tibshirani et al., 2002) (see Appendix K Section K.1.2) were used. Figure 5.7.1 shows the elbow plot used to determine the optimal number of clusters. Figure 5.7.2 shows the average silhouette score plotted against the number of clusters. Lastly, Figure 5.7.3 shows the gap statistic.

From Figures 5.7.1 and 5.7.2, it can be seen that according to the elbow method and the average silhouette score, the optimal number of clusters is 3. On the other hand, from Figure 5.7.3, it can be seen that according to the gap statistic, the optimal number of clusters is 1. However, selecting $K = 1$ is not meaningful in this context, as

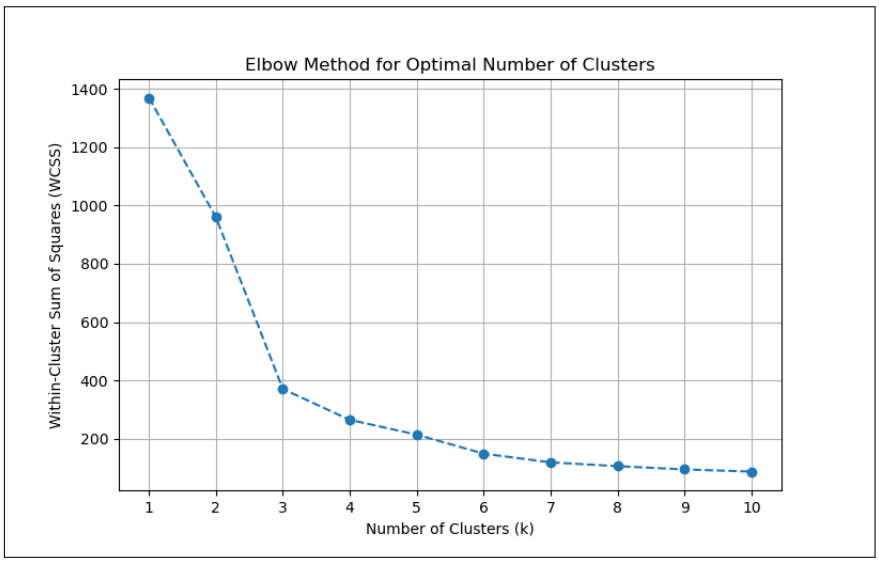

Figure 5.7.1: Elbow plot used to determine the optimal number of clusters.

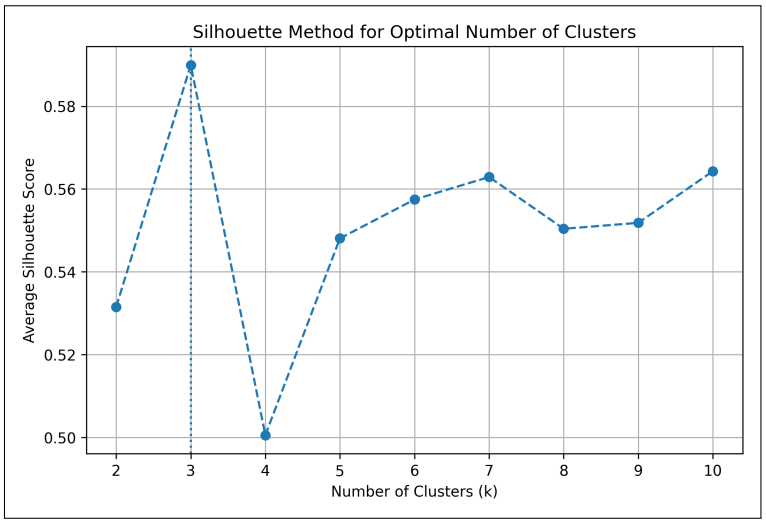

Figure 5.7.2: Average Silhouette score plot used to determine the optimal number of clusters.

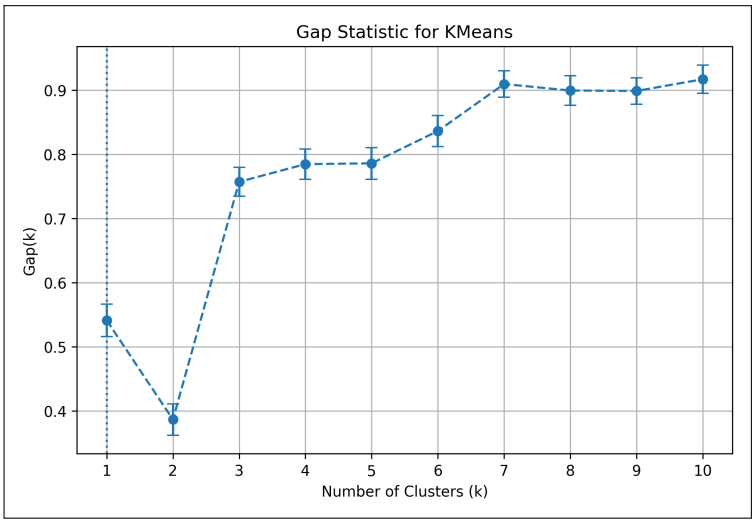

Figure 5.7.3: Gap statistic plot used to determine the optimal number of clusters.

it would imply that the entire dataset constitutes a single cluster, which is the same as fitting the two architectures on the original dataset (see Sections 5.5 and 5.6). Since two out of the three methods identify $K = 3$ as optimal and $K = 1$ is trivial, it was decided to apply K-means with the number of clusters set to 3. The clustering obtained using K-means is illustrated in Figure 5.7.4.

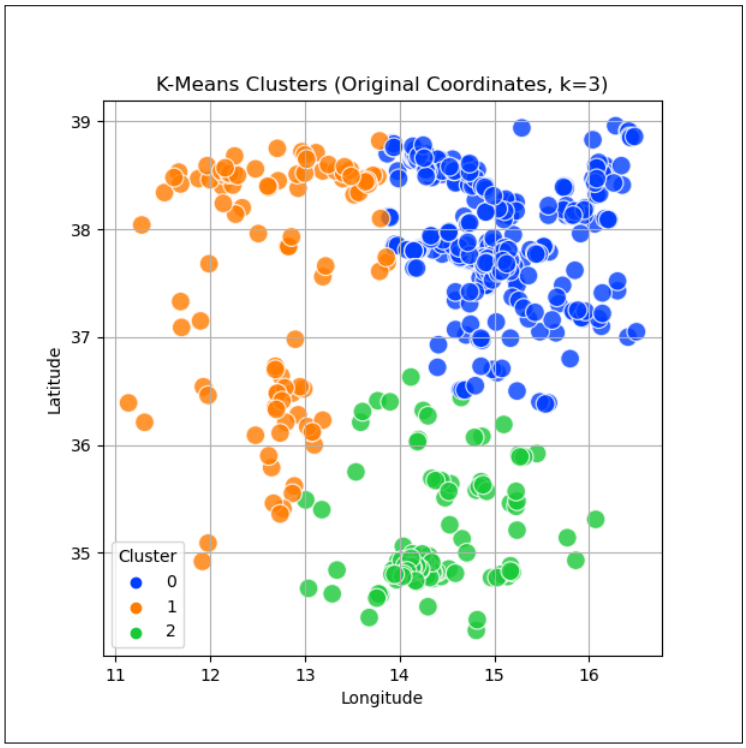

Figure 5.7.4: Clustering obtained using K-means.

Subsequently, DBSCAN clustering was applied. DBSCAN takes two parameters, namely `MinPts` and `Eps`. The former is specified according to how many variables are being considered. In the case of this application, two variables, namely, latitude and longitude of the events, are being considered for the clustering part. According to the literature, when the number of variables is 2, `MinPts` is conventionally set to 5 (Ester et al., 1996). The optimal value of `Eps` is selected using an elbow plot, where the "elbow" is selected as the optimal value for `Eps`. Figure 5.7.5 shows the elbow plot used to determine the optimal value for `Eps`.

From Figure 5.7.5, it can be concluded that the optimal value for `Eps` is 0.3. Therefore, DBSCAN with `MinPts`=5 and `Eps`=0.3 was implemented. The clustering resulting from DBSCAN can be found in Figure 5.7.6, where red points, labelled as -1, are indicative of outliers.

As shown in Figure 5.7.6, the clustering produced by DBSCAN differs from that of K-means. DBSCAN yielded fewer clusters with most points being grouped into

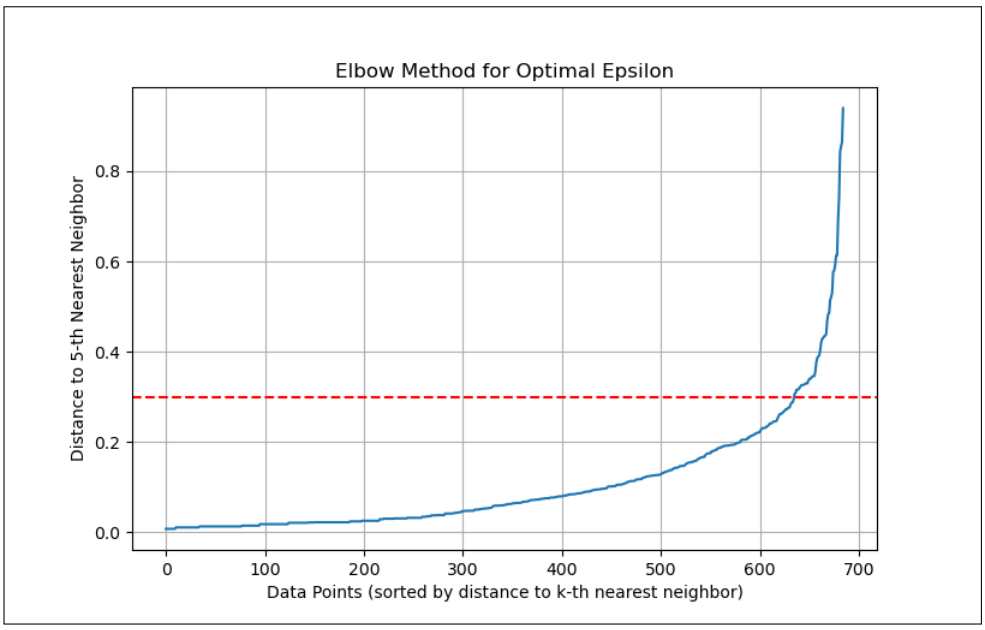

Figure 5.7.5: Elbow plot used to determine the optimal `Eps`.

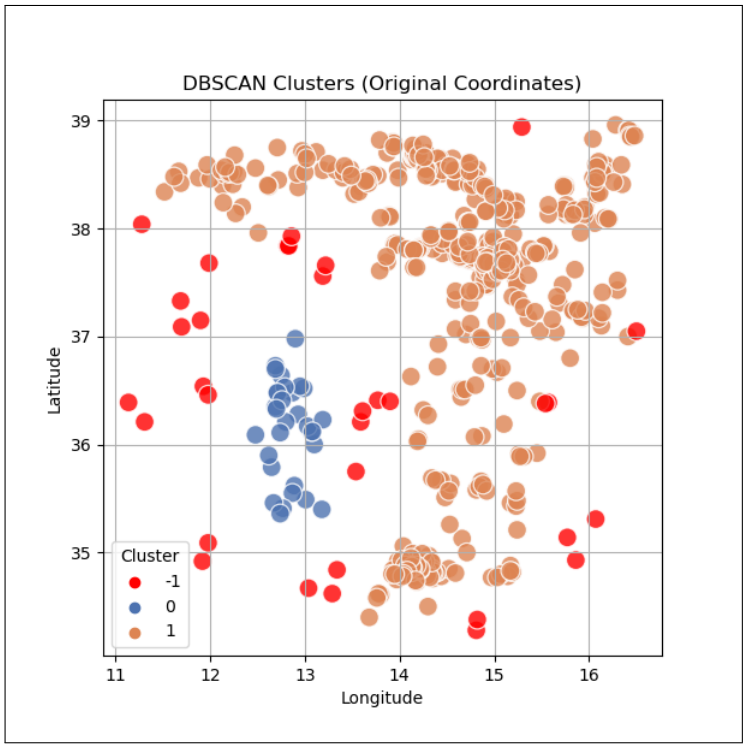

Figure 5.7.6: Clustering obtained using DBSCAN.

one large cluster, and the remainder either in a small secondary cluster or labelled as outliers. However, because DBSCAN relies on a single global density threshold, it can struggle to detect clusters when densities vary, which may be the case in our data. Thus, it was decided to also apply HDBSCAN to our data.

HDBSCAN has two main parameters, specifically, `min_cluster_size` and `min_samples`. The former represents the smallest size grouping that HDBSCAN will consider a valid cluster, with larger values leading to fewer, larger clusters and smaller

values resulting in smaller, more detailed clusters. The latter controls how conservative the clustering is. It sets how many points are needed around a point for it to avoid being marked as outliers. Higher values make the algorithm more robust to outliers but also more likely to label sparse regions as outliers. A grid search method was used to pick best parameters for HDBSCAN, evaluated on the respective modified Dunn's index (Ilc, 2012). The modified Dunn's index is a validity metric used to compare different resulting clusterings, taking into account that the data may contain non-convex shaped clusters (Ilc, 2012). The highest modified Dunn index suggests the best parameter configuration. More information on the modified Dunn index can be found in Appendix K, Section K.5. The values considered for `min_cluster_size` were $\{50, 70, 100\}$ and the values considered for `min_samples` were $\{5, 10, 20, 30\}$. In total, there were 12 combinations and the grid search results can be found in Table 5.7.1.

`min_cluster_size`	`min_samples`	**No. of Clusters**	**Modified Dunn Index**
50	30	2	3.39
70	30	2	3.39
100	30	2	3.39
50	20	3	3.27
100	10	2	2.98
70	20	2	2.95
100	20	2	2.95
100	5	2	2.92
50	10	3	2.15
70	10	3	2.15
50	5	3	1.95
70	5	3	1.95

Table 5.7.1: Modified Dunn Index of different HDBSCAN parameter configurations.

From Table 5.7.1 it can be concluded that according to the modified Dunn's index the optimal value for `min_samples` is 30. With regards to `min_cluster_size`, the best values are $\{50, 70, 100\}$. It was decided to select 50 as `min_cluster_size` since it is the smallest value that achieves the largest modified Dunn's index, and so, the finest-grained partitioning is achieved without any loss in cluster quality. This higher granularity can reveal more detailed subregions while maintaining the same overall compactness and separation. Thus, HDSBCAN with parameter configuration `min_samples`=30 and `min_cluster_size`=50 was implemented, and the clustering result can be found in Figure 5.7.7. Once again, outliers are presented by red points and labelled as -1.

Figure 5.7.7 shows that the HDBSCAN method resulted in two clusters with most points being classified as outliers. The clustering obtained differs significantly from that of DBSCAN. When compared to the clustering obtained from K-means,

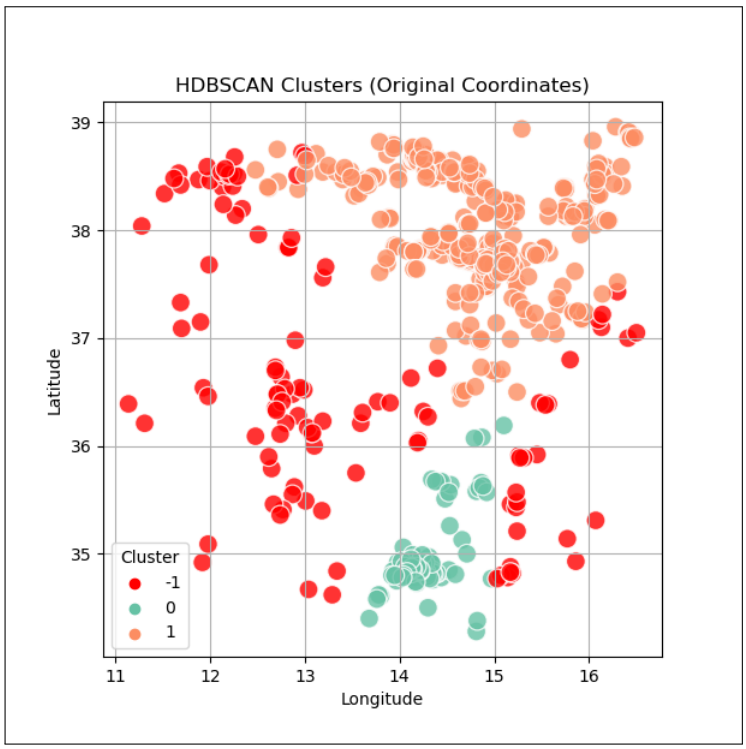

Figure 5.7.7: Clustering obtained using HDBSCAN.

it is observed that the top right cluster was identified by both methods. The bottom right cluster was identified by both methods, however, the one identified by HDBSCAN is notably smaller and more dense. Lastly, the left cluster identified by K-means was considered as outliers by HDBSCAN. Nevertheless, an empirical method for comparing the clustering quality is required. In order to empirically determine which clustering algorithm produced the best partition, the modified Dunn's index was calculated for each partition and tabulated in 5.7.2.

Clustering Method	**Modified Dunn's Index**
K-Means	1.90
DBSCAN	1.39
HDBSCAN	4.23

Table 5.7.2: Modified Dunn's index for the partitions produced by the different clustering algorithms.

Upon inspecting Table 5.7.2 it can be concluded that the best clustering method is HDBSCAN as its score was the largest. Following this, the data points were mapped back to their original scale, and plotted on a world map to view the geographical locations of the HDBSCAN clusters. This can be seen in Figure 5.7.8.

Considering the results obtained, it was decided to re-do the experiments conducted in Sections 5.5 and 5.6 on HDBSCAN Cluster 1. This is because it had the largest sample size, which is an important consideration when working with statistical learning models. In total there were 416 events recorded by 52 broadband seismic sta-

tions. Maps of events and stations of the region represented by HDBSCAN Cluster 1 are illustrated in Figures 5.7.9 and 5.7.10.

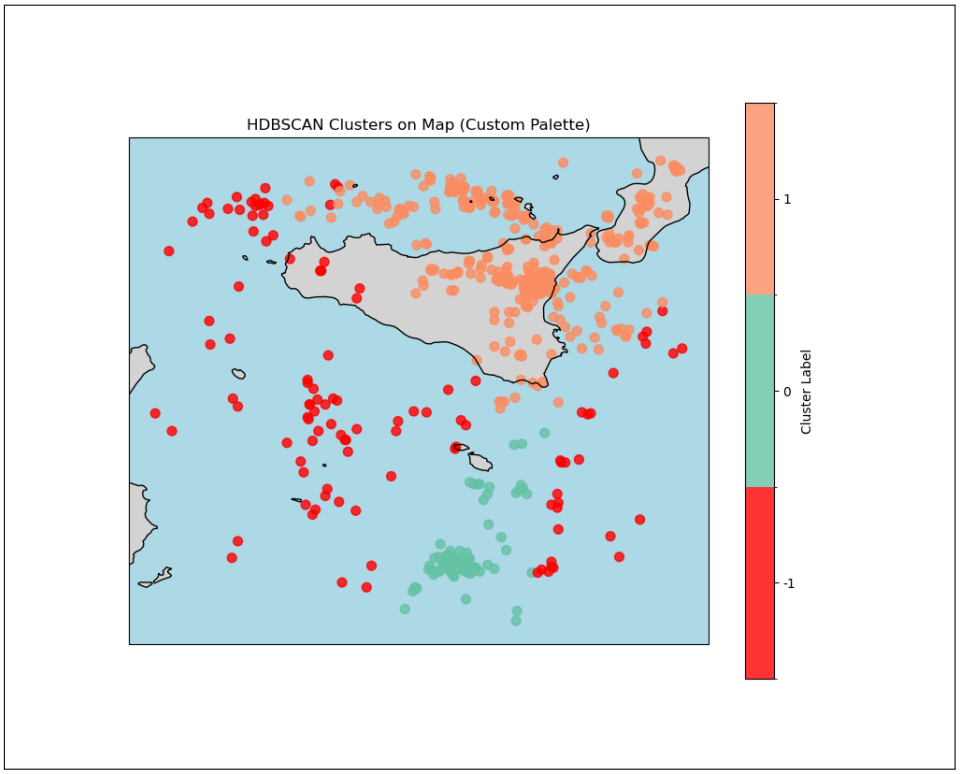

Figure 5.7.8: Clustering obtained using HDBSCAN illustrated on a map.

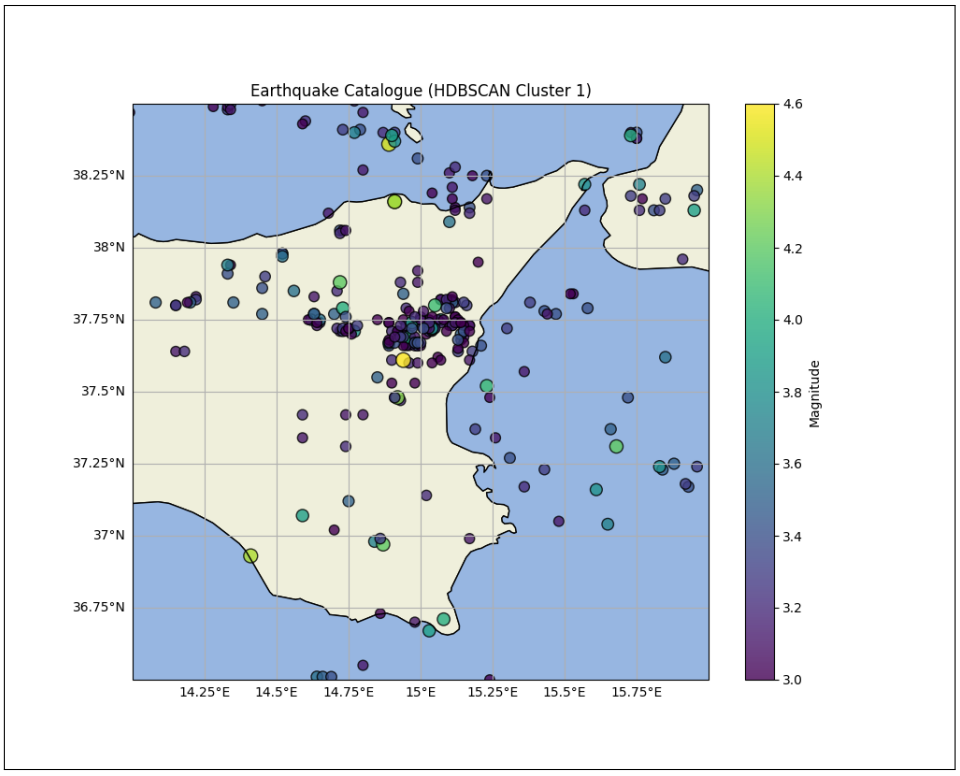

Figure 5.7.9: Map of seismic events in HDBSCAN Cluster 1 between January 2013 and December 2024.

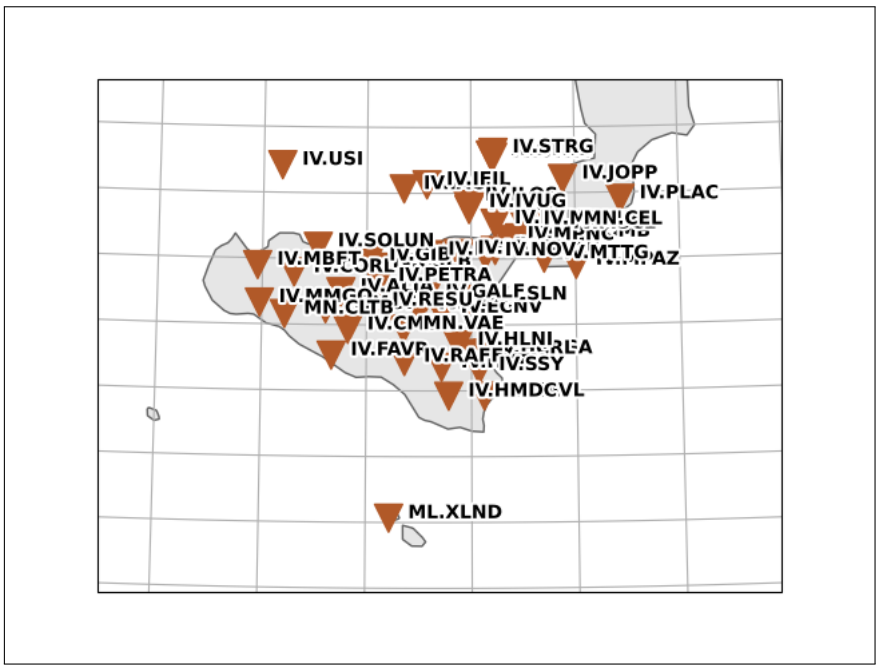

Figure 5.7.10: Map of the seismic stations recording information on the seismic events in HDBSCAN Cluster 1 between January 2013 and December 2024.

5.7.1 Edgeless Graph Model Architecture

As in Section 5.5, we first performed hyperparameter tuning to identify optimal settings for the edgeless graph architecture, this time restricting training and validation to events in HDBSCAN Cluster 1 rather than the full dataset. The full search required approximately 14.5 hours. A comprehensive summary of validation metrics is provided in Appendix L, Tables L.1-L.10. Initially, we planned to select individual model configurations for each target variable, and an overall configuration, purely on the basis of validation metrics MAE, MSE and NRMSE, before examining loss and prediction plots. However, inspection of Tables L.1-L.10 showed that most configurations yielded near identical validation metrics. This could be attributed to the fact that model fitting and evaluation were conducted on a single, low-variance cluster, which encourages many hyperparameter settings to converge toward mean-biased solutions. The effect is particularly pronounced for latitude, longitude, and magnitude, whose distributions within Cluster 1 are tightly concentrated (limited geographic spread and a narrow magnitude range), so predictors that regress toward the cluster mean achieve similar average errors. Consequently, we prioritized loss curves and predicted versus actual target values plots to select the best configuration based on training dynamics, generalization ability and accuracy. All plots for this section are available in the Google Drive repository referenced in Appendix E (folder Clustered_Edgeless Graph Model). The top four configurations for the edgeless architecture on clustered data are reported

in Table 5.7.3.

Model	Batch Size	LR	Dropout Prob
Model 7	32	0.001	0.2
Model 8	64	0.001	0.2
Model 9	16	0.001	0.2
Model 10	64	0.0001	0.2

Table 5.7.3: Best hyperparameter configurations for Edgeless Graph Architecture on HDSBCAN Cluster 1.

From Table 5.7.3, the optimal configurations are very similar; the only substantive difference among Models 7-9 is the batch size, while Model 10 differs by using a smaller learning rate. Inspection of the loss curves (see Google Drive link in Appendix E) yields three consistent observations. Firstly, runs with lower learning rates, for example, $1e-5$ exhibit slower convergence and more pronounced oscillations. Secondly, higher learning rates, such as 0.01, often show rapid initial drops in loss. Larger steps help the optimizer escape flat regions and reach a better basin early, but can also introduce implicit regularization that biases predictions toward central values, which manifests as tighter horizontal bands in the prediction plots. Conversely, the lower learning rate tends to yield larger error bars and occasional under/overestimation, reflecting incomplete convergence. Lastly, at very low ($1e-5$) or very high (0.01) learning rates, varying dropout had little effect on either losses or predictions, however, at intermediate learnings rates, such as 0.001 and 0.0001, higher rates of dropout degraded model training/validation performance.

The loss plots for Models 7-10 can be found in Figures H.2-H.5. Comparing Figures H.2-H.5, it can be seen that in terms of training dynamics Model 8 is the best model. This is because the loss curves are notably the smoothest, with the mild oscillations being attributed to normal stochastic variation. Furthermore, it converges the fastest with an early plateau, and it has the smallest, most stable train-validation gap suggesting the evidence of the best generalization. After this Models 7 and 9 rank the best as they contain slight oscillations and a slightly larger train-validation gap when compared to Model 8. Lastly, Model 10 is the worst because it has the most oscillations, specifically in the training loss, and the train-validation curves are poorly aligned, with validation loss persistently below training loss, a pattern consistent with underfitting or overly conservative learning.

The predicted versus actual target values plots for Models 7-10 can be found in Figures H.10-H.17. Upon inspecting Figures H.10-H.17 it can be seen that across the four configurations, Model 8 provides the strongest generalization ability and accuracy.

This is because its train-validation gaps are consistently smaller, and the scatter aligns closely with the diagonal for latitude and longitude, with generally small and uniform error bars. Although depth remains challenging, the spread is controlled. Model 7 is a close second as it aligns the diagonal somewhat well, particularly for depth, but the train-validation gaps are larger, suggesting slightly weaker generalization than Model 8. Model 9 performs adequately but exhibits wider error bars and less consistent alignment with the diagonal, indicating more variance and mild underfitting/overfitting trade-offs across targets. Lastly, Model 10 is the worst performing model, as it shows horizontal banding, larger error bars, and higher validation error, despite competitive train-validation gaps in some variables.

Taking everything into consideration, it was decided to pick Model 8 with parameter settings batch size = 64, learning rate = 0.001 and dropout probability = 0.2 as the best for the edgeless graph architecture on HDBSCAN Cluster 1 data. This is because it exhibited the best training dynamics, as observed from the loss plots, as well as the best generalization ability and accuracy, as seen by the predicted versus actual target values plots. It is worth noting that the loss curves for Model 8 seem to plateau around the 200^{th} epoch with very little improvement afterwards. This suggests that extending training beyond 250 epochs yields diminishing returns, and the model could benefit from early stopping.

Subsequently, early stopping was applied to Model 8, setting `patience` = 20 epochs. This model is referred to as Model 11. Figure H.6 shows the loss plot for Model 11. Furthermore, Figures H.18 and H.19 illustrate the predicted versus target values. Notably, training stopped after 151 epochs, with the best weights restored from epoch 131, as determined by the early stopping criterion.

When comparing the loss plots in Figures H.3 and H.6, it can be seen that Model 11 shows more stable convergence with training and validation losses closely aligned, indicating better generalization and reduced risk of overfitting compared to the second model. However, upon examining Figures H.18 and H.19, it can be observed that Model 11's predictions are significantly worse than Model 8's. This is because the data points for all target variables form a horizontal band, and the error bars are substantially large for all target variables, hinting at great uncertainty. Taking everything into consideration it can be concluded that adding early stopping to our model, degrades model performance. As a result, it was decided to retain Model 8 as the best model for the edgeless graph architecture applied to the reduced dataset. A summary of the

validation metrics of Model 8 can be found in Table 5.7.4.

Variable	MAE	MSE	NRMSE
Latitude	0.38	0.22	1.02
Longitude	0.46	0.44	1
Depth	6.51	58.53	0.94
Magnitude	0.28	0.14	1.09
All			1.01

Table 5.7.4: Summary of Validation Performance Metrics for Model 8: batch size=64, learning rate=0.001, dropout probability=0.2.

5.7.2 Dynamic Edges GNN Model Architecture

As in Section 5.6, we first performed hyperparameter tuning to identify optimal settings for the dynamic edges GNN model architecture, this time restricting training and validation to events in HDBSCAN Cluster 1 rather than the full dataset. The full search required approximately 24.83 hours. A comprehensive summary of validation metrics is provided in Appendix M, Tables M.1-M.15. As discussed in Section 5.7.1, candidate models trained and validated on a single, low-variance cluster produce near-identical validation metrics. Accordingly, we adopt the same criterion and prioritize training dynamics, generalization ability and accuracy, using loss-curve behaviour and predicted-versus-actual target values plots to identify the best configuration. The complete set of plots is available in the repository cited in Appendix E (see folder Clustered_Dynamic Edges GNN Model), and the top three configurations for the dynamic-edges GNN architecture fitted on HDBSCAN Cluster 1 data are reported in Table 5.7.5.

Model	k	Batch Size	LR	Dropout Prob
Model 12	3	16	0.001	0.2
Model 13	3	64	0.001	0.2
Model 14	4	32	0.001	0.2

Table 5.7.5: Best hyperparameter configurations for Dynamic Edges GNN Architecture on HDSBCAN Cluster 1.

From Table 5.7.5, the optimal configurations are very similar with all configurations having learning rate 0.001 and dropout probability 0.2. The loss plots for Models 12-14 can be found in Figures J.4-J.6. From Figures J.4-J.6 it can be seen that the configuration that exhibited the best training behaviour is Model 13. This is because its curves are the smoothest, convergence has been reached at around epoch 150, and it has the smallest train-validation gap. Model 14 ranks a close second since aside from the brief initial early spikes, it exhibits similar training dynamics to Model 13. However, it is worth noting that Model 14 has slightly more oscillations. Model 12 ex-

hibits the worst training behaviour since although it converges quickly after the early drop, its validation loss begins to drift upward sooner and the train-validation gap widens steadily, reflecting weaker generalization. Overall, Model 13 offers the best balance of smoothness, convergence speed, and generalization, with Model 14 a credible alternative and Model 12 the least favourable.

The predicted-versus-actual target values plots for Models 12-14 can be found in Figures J.15-J.20. From Figures J.15-J.20 it can be seen that among the three configurations, Model 12 exhibits the weakest generalization ability and the lowest accuracy. This is because the train-validation error gaps are the largest and the scatter is substantial, particularly for depth and magnitude, indicating poor generalization. Model 13 and 14 are comparable with Model 13 performing better for latitude and longitude, and Model 14 performing better for depth and magnitude. However, Model 13 shows some horizontal banding and slightly larger error bars. This suggests Model 14 has the best generalization ability and accuracy. Since the differences in loss curves are minor but the prediction gains are greater, we select Model 14 as the optimal configuration.

Looking at Figure J.6, it can be seen that Model 14 could benefit from early stopping as the validation loss begins to increase after approximately 180 epochs while the training loss continues to decrease, indicating overfitting. Thus, it was decided to refit Model 14 using early stopping with a patience of 20, and this refitted version will be referred to as Model 15. Figure J.7 shows the loss plot for Model 10, while Figures J.21 and J.22 illustrate the predicted versus target values. Notably, training stopped after 27 epochs, with the best weights restored from epoch 7, as determined by the early stopping criterion.

When comparing the loss plots in Figures J.6 and J.7, it can be seen that Model 15 seems to converge faster and maintains a smaller gap between training and validation loss towards the end, showing better generalization, however, it is noisier. Conversely, Model 14 has smoother loss curves and achieves a lower training loss, but converges slowly and shows mild overfitting after extended epochs.

Additionally, upon examining Figures J.21 and J.22, it can be observed that Model 15's training and validation predictions are relatively worse than Model 14's. This is because the data points for all target variables form a horizontal band, and for variables latitude, longitude and magnitude are completely out of range. Furthermore, the error bars are substantially large for all target variables, hinting at great

uncertainty. Also, when compared to Figures J.19 and J.20, the MAE and MSE values associated with Model 15 are slightly larger than those of Model 14. Taking everything into consideration it can be concluded that adding early stopping to our model, degrades model performance. As a result, it was decided to retain Model 14 as the best model for the dynamic edges graph architecture applied to the reduced dataset. A summary of the validation metrics of Model 14 can be found in Table 5.7.6.

Variable	MAE	MSE	NRMSE
Latitude	0.46	0.34	1.26
Longitude	0.51	0.53	1.01
Depth	6.65	71.68	1.05
Magnitude	0.26	0.13	1.05
All			1.11

Table 5.7.6: Summary of Validation Performance Metrics for Model 14: $k = 4$, batch size=32, learning rate=0.001, dropout probability=0.2.

5.8 Model Comparison

Having presented the results obtained from the experimental analysis and identified the best-performing model within each architecture, this section proceeds to provide a direct comparison between the two architectures. Moreover, through ensemble modelling we attempt to improve predictive accuracy of the models. In addition, the impact of applying the architectures to the reduced dataset is also examined. Finally, the optimal models from each architecture are applied to independent test data to assess their forecasting ability.

5.8.1 Comparison of Edgeless Graph Architecture and Dynamic Edges GNN Architecture

In Sections 5.5 and 5.6 it was seen that on the original dataset (see Section 5.3), the best models for the edgeless graph architecture and the dynamic edges GNN architecture were Model 3 (batch size = 32, learning rate = 0.001 and dropout probability = 0.2 using early stopping setting `patience`=20) and Model 4 ($k = 3$, batch size = 32, learning rate = 0.001 and dropout probability = 0.4), respectively. Table 5.8.1 summarizes the validation metrics for Model 3 and 4. Comparing the validation metrics in Table 5.8.1, it can be observed that the metrics for Model 3 and 4 are comparable, with Model 3 slightly outperforming Model 4 in latitude and Model 4 slightly outperforming Model 3 in depth. This suggests that in terms of validation metrics, Model 3 and Model 4 have similar performance.

Examining the loss plots for Model 3 and Model 4, which are in Figures H.1 and

	Model 3			Model 4		
Variable	MAE	MSE	NRSME	MAE	MSE	NRSME
Longitude	0.78	1.19	0.76	0.88	1.48	0.84
Latitude	0.64	0.79	0.99	0.62	0.79	0.99
Depth	6.31	55.03	1.01	6.44	56.63	1.027
Magnitude	0.27	0.12	0.98	0.28	0.12	0.98
All			0.94			0.96

Table 5.8.1: Validation performance metrics for Edgeless Graph Architecture using best hyperparameter configuration (Model 3) and Dynamic Edges GNN Architecture using best hyperparameter configuration (Model 4)

J.1, respectively, it can be seen that both models exhibit good training dynamics. They both converge well, show only minor oscillations, and main little to no train-validation gap, indicating good generalization. The predictions for Model 3 are found in Figures H.7 and H.8, whereas for Model 4, they are found in Figures J.8 and J.9. With regards to Model 3, predictions for latitude cluster somewhat close to the 1:1 diagonal. For longitude this is less apparent and there is some horizontal banding present. Depth predictions align somewhat with the 1:1 line, but demonstrate noticeable underestimation at higher depths. Magnitude is fairly close to the 1:1 line, but slightly biased low at large values. Model 4's predictions for longitude and magnitude are somewhat comparable to those of Model 3 in terms of closeness to the diagonal line. However, with regards to latitude and depth, the predictions are farther away from the 1:1 line when compared with those of Model 3. With regards to the gaps between training/validation MAEs and MSEs, it seems that Model 3 has smaller gaps overall. Lastly, Model 3 has smaller error bars than Model 4, indicating lower model uncertainty. Overall, it seems Model 3 generalizes better and provides more consistent predictions across all variables.

Taking everything into account, although Models 3 and 4 achieved comparable validation metrics and training dynamics, Model 3 is deemed to be the better model because its predicted-versus-actual target values plots demonstrated better model fit, smaller train/validation metric gaps and provides more consistent generalization across variables.

It is worth mentioning the reason as to why larger depths and magnitudes might be underestimated is likely due to their under-representation in the dataset, since the central Mediterranean region records relatively few high-magnitude events. The observed over- and underestimation in latitude and longitude can also be explained by the uneven distribution of seismic stations and events across the study area. As seen in Figures 5.3.2 and 5.3.3, many events occur in regions that are sparsely instrumented,

and because the model relies on station coordinates for location predictions, this uneven coverage introduces systematic biases.

When comparing our results with similar studies applied to other regions, such as California (Van den Ende & Ampuero, 2020; X. Zhang et al., 2022), their models achieve slightly better performance, benefitting from larger datasets, denser station coverage, and events clustered close to instrumentation. Nevertheless, even these studies continue to face challenges in accurately predicting depth, highlighting the general difficulty of this task (Van den Ende & Ampuero, 2020; X. Zhang et al., 2022).

5.8.2 Ensemble Modelling

To determine whether predictive accuracy on training and validation sets could be improved beyond the single best models, we adopted an ensemble approach (Sengupta et al., 2020). Ensemble methods are commonly used to improve generalisation performance by combining multiple well-performing models with the aim of stabilising predictions and decreasing sensitivity to individual model configurations. In this work, all candidate models were trained on the same dataset (the full dataset), but had different hyperparameter settings, leading to multiple competitive models with slightly different representations. For the edgeless graph architecture (Section 5.5) and the dynamic-edges GNN architecture (Section 5.6), we selected the top five configurations by overall NRMSE and formed ensembles by averaging their point predictions. This design choice balances predictive robustness and computational cost while retaining only the most accurate models. We then computed training and validation NRMSE per target variable and overall for each ensemble, and inspected prediction-versus-actual plots. The ensemble predictions for the edgeless graph and dynamic-edges GNN are shown in Figures H.20 and J.23, respectively. Recall that the best edgeless graph model is Model 3, while the best dynamic edges GNN model is Model 4. The predictions on the training and validation sets for Model 3 and 4 are found in Figures H.9 and J.10, respectively.

Comparing the two ensembles (see Figures H.20 and J.23), overall train-validation NRMSEs and scatter are broadly comparable for latitude, longitude, and magnitude; the edgeless graph ensemble is modestly better for depth, tracking the diagonal more closely. Against their respective single-model counterparts, however, both ensembles underperform: Model 3 attains lower train-validation NRMSEs and closer alignment to the diagonal than the edgeless graph ensemble, and Model 4 similarly surpasses the dynamic edges GNN ensemble in both metrics and calibration.

In conclusion, ensembling worsened accuracy in this setting. The likely reason is that the candidate models are highly similar, so averaging not only reduces variance only marginally, but also increases bias towards the mean. This effect is visible in the horizontal banding. In short, the ensembles dilute the strengths of the best model while introducing additional bias, leading to poorer overall performance.

5.8.3 Impact of Reduced Dataset

In Section 5.7, it was observed that seismic activity in the central Mediterranean region is unevenly distributed, with events and stations clustering in localized pockets. Consequently, clustering was conducted to find hidden groups in the data, and the experiments in Sections 5.5 and 5.6 were re-conducted on the reduced dataset with the aim of enabling more focused models that can improve localization accuracy and efficiency. In this section, any changes that arose as a result of reducing the dataset will be discussed. First, the difference between applying the architectures to the original dataset and the reduced dataset will be outlined. This will be followed by a comparison of the optimal edgeless graph model and the dynamic edges GNN model on the reduced dataset.

From Section 5.7.1 it was seen that the best edgeless graph model on the reduced dataset was Model 8 (batch size = 64, learning rate = 0.001 and dropout probability= 0.2). Table 5.8.2 summarizes the validation metrics for Model 3 and 8. Comparing the validation metrics in Table 5.8.2, for magnitude and the overall NRMSE, performance metrics are comparable with slight differences. For latitude and longitude, the MAE and MSE are significantly lower for Model 8, but the NRMSE is also somewhat comparable. Conversely, Model 3 attained overall lower validation metrics for depth. Upon inspecting the loss plots of Model 3 and Model 8 in Figures H.1 and H.3, respectively, it can be seen that Model 3 has better training behaviour that Model 8 since Model 8 has a larger train-validation loss gap and does not converge as well as Model 3. With regards to the prediction plots for Model 3 (see Figures H.7 and H.8) and Model 8 (see Figures H.12 and H.13), it can be observed that the overall predictions for Model 8 are closer to the 1:1 line with less scatter and smaller error bars. Hence, applying the edgeless graph architecture to the reduced dataset resulted in better model performance.

From Section 5.7.2 it was seen that the best dynamic edges GNN model on the reduced dataset was Model 14 ($k = 4$, batch size = 32, learning rate = 0.001 and dropout probability = 0.2). Table 5.8.3 summarizes the validation metrics for Model 4

	Model 3			Model 8		
Variable	MAE	MSE	NRSME	MAE	MSE	NRSME
Longitude	0.78	1.19	0.76	0.38	0.22	1.02
Latitude	0.64	0.79	0.99	0.46	0.44	1
Depth	6.31	55.03	1.01	6.51	58.53	0.94
Magnitude	0.27	0.12	0.98	0.28	0.14	1.09
All			0.94			1.01

Table 5.8.2: Validation performance metrics for Edgeless Graph Architecture using best hyperparameter configuration on Full Dataset (Model 3) and Edgeless Graph Architecture using best hyperparameter configuration on Reduced Dataset (Model 8)

and 14. Comparing the validation metrics in Table 5.8.3, for magnitude and the overall NRMSE, performance metrics are comparable with slight differences. For latitude and longitude, the MAE and MSE are significantly lower for Model 14, but the NRMSE is also somewhat comparable. Conversely, Model 4 attains lower validation metrics for depth. Upon inspecting the loss plots of Model 4 and Model 14 in Figures J.1 and J.6, respectively, it can be concluded that Model 4 shows a smaller train-validation loss gap, better convergence and smoother loss curves. With regards to the prediction plots for Model 4 (see Figures J.8 and J.9) and Model 14 (see Figures J.19 and J.20), it can seen that Model 14 produces better predictions overall as data points lie closer to the 1:1 line and error bars are smaller. On the other hand, Model 4 has smaller train/validation metric gaps. However, this might be attributed to the larger sample size. Thus, although Model 4 exhibits smoother loss curves and smaller training/validation loss gaps, Model 14 produced more accurate predictions across variables, highlighting better generalization ability despite nosier training dynamics. Hence, applying the dynamic edges GNN architecture to the reduced dataset resulted in better model performance.

	Model 4			Model 14		
Variable	MAE	MSE	NRSME	MAE	MSE	NRSME
Longitude	0.88	1.48	0.84	0.46	0.34	1.26
Latitude	0.62	0.79	0.99	0.51	0.53	1.01
Depth	6.44	56.63	1.027	6.65	71.68	1.05
Magnitude	0.28	0.12	0.98	0.26	0.13	1.05
All			0.96			1.11

Table 5.8.3: Validation performance metrics for Dynamic Edges GNN Architecture using best hyperparameter configuration on Full Dataset (Model 4) and Dynamic Edges GNN Architecture using best hyperparameter configuration on Reduced Dataset (Model 14)

Comparing Models 8 and 14, it can be seen that with regards to performance metrics, Model 8 achieved lower values overall. Moreover, Model 8 has smoother loss curves, with smaller training/validation loss gaps and better convergence. Furthermore, with regards to predictions, Model 8 provides a more balanced and generalizable model across most variables. This suggests that even on the reduced dataset, the edgeless graph architecture outperforms the dynamic edges GNN architecture.

5.8.4 Evaluation on Independent Test Set

In order to assess the predictive ability of Models 3 and 4, it was decided to apply them on a test dataset. For the test dataset, the same geographic parameters were set, however, this time only events between 1st January 2025 to 26th August 2025 were considered. The test data contains 26 total events recorded by 34 seismic stations, which can be seen in Figures 5.8.1 and 5.8.2, respectively. Note that this sample size is considered to be rather small for the statistical learning framework, however, no other data was available. Model performance and predictive ability will be assessed by examining the test loss, as well as the performance metrics MAEP, MSEP and NRMSEP presented in Section 2.7. Additionally, the predicted-versus-actual target values plots, generated using test data, for each model will also be examined. It is worth noting that an independent test set could not be created for the reduced dataset, as 26 events already represent a small sample, and further division would have made evaluation even more unreliable.

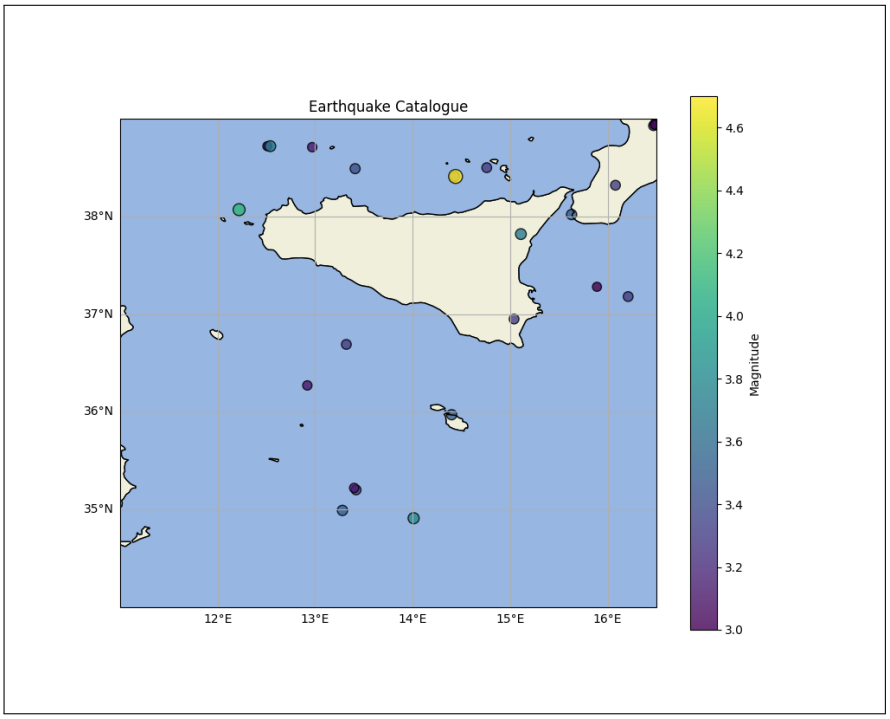

Figure 5.8.1: Map of the seismic events in the central Mediterranean region between 1st January 2025 to 26th August 2025.

The test loss for Model 3 and Model 4 are 0.046 and 0.05, respectively, suggesting that Model 3 outperforms Model 4. The performance metrics for Model 3 and Model 4 can be found in Table 5.8.4, respectively. Additionally, the predicted-versus-actual target values plots for Models 3 and 4, can be found in Figures 5.8.3 and 5.8.4.

Upon inspecting Figures 5.8.3 and 5.8.4, it is difficult to determine which model

Figure 5.8.2: Map of seismic stations recording information on the seismic events in the central Mediterranean region between 1st January 2025 to 26th August 2025.

	Edgeless Graph Model			dynamic edges GNN model		
Variable	MAEP	MSEP	NRSMEP	MAEP	MSEP	NRSMEP
Longitude	2.09	6.94	1.2	2.32	7.83	1.27
Latitude	1.16	1.77	1	1.27	1.96	1.06
Depth	7.16	72.07	0.99	7.7	74.56	1
Magnitude	0.26	0.13	0.98	0.3	0.14	1.01
All			1.05			1.09

Table 5.8.4: Test performance metrics for Edgeless Graph Architecture using best hyperparameter configuration and Dynamic Edges GNN Architecture using best hyperparameter configuration

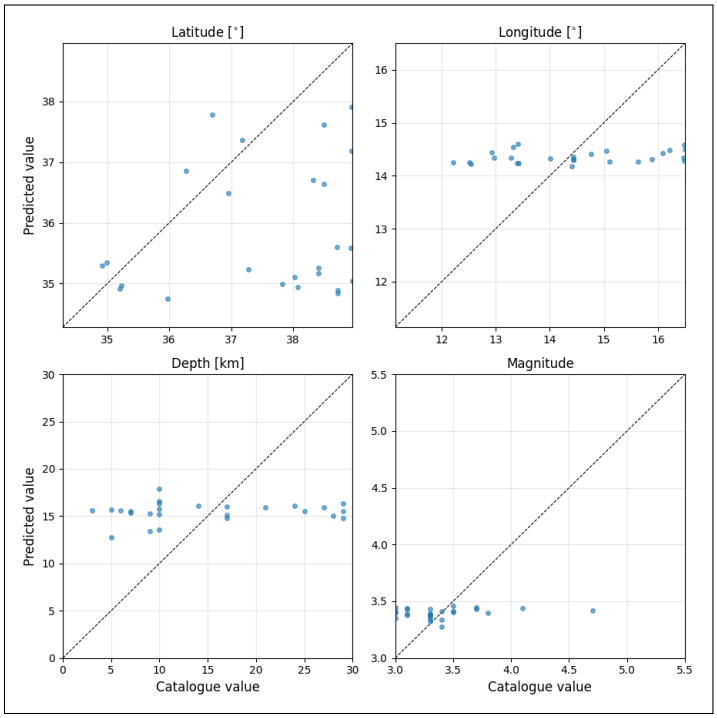

Figure 5.8.3: Predictive plots for Edgeless Graph Model on test data.

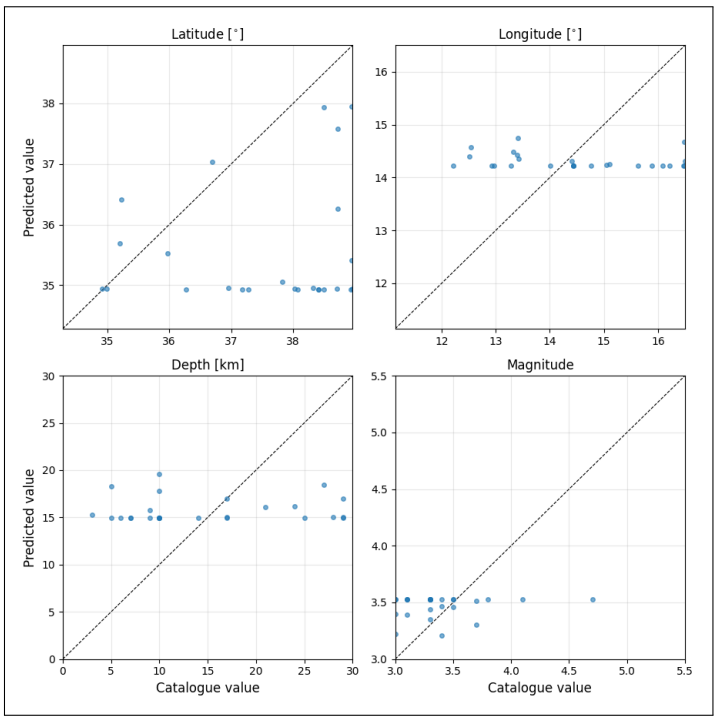

Figure 5.8.4: Predictive plots for dynamic edges GNN model on test data.

performs best, as both plots show inconsistencies, likely attributed to the relatively small and imbalanced nature of the test set. Consequently, the evaluation will be based on the test performance metrics for a more objective comparison. From Table 5.8.4 it can be observed that the edgeless graph model (Model 3) achieved lower performance metrics than the dynamic edges GNN model (Model 4), and so, Model 3 is considered to be the best model in terms of predictive ability.

Earthquake source characterization in the Central Mediterranean region remains relatively understudied compared to well-instrumented areas such as Japan and California. In this context, Model 3 provides one of the first baselines for future studies in the region. Beyond serving as a benchmark, the model could also be used for transfer learning, enabling researchers to adapt its trained representations to similar future studies.

In addition to establishing a baseline, Model 3 has revealed systematic biases in earthquake source characterization. The analysis shows that large-magnitude events are more difficult to predict, primarily due to their limited number in the dataset. Moreover, shallow, onshore crustal earthquakes are generally better resolved than offshore or deeper events. These systematic variations are meaningful, as they highlight both the challenges and limitations of monitoring earthquakes in the Central Mediterranean region. At the same time, they can guide researchers toward potential improvements. For example, station coverage in poorly represented regions can be expanded in con-

strained offshore areas or the density of stations where wave propagation is strongly affected by heterogeneous crustal structures can be increased.

Although the predictions from Model 3 are not perfectly accurate, they still provides practical value. Rapid, first-order estimates of earthquake location, depth, and magnitude can support preliminary source characterization. Traditional inversion methods, which are widely used to characterize earthquake sources, are iterative and non-linear in nature, and can converge slowly or to a local minimum if initialized with poor starting parameters (Ma et al., 2022). Using Model 3's estimates as a "smart" initial guess can therefore reduce computational cost and accelerate convergence. Furthermore, the model has potential as a pre-processor within hazard assessment and early warning systems, providing approximate real-time estimates that can trigger downstream analyses while more refined inversions are underway.

Chapter 6

Concluding Remarks

6.1 Summary of Dissertation

This dissertation set out to investigate the use of NNs for seismic source characterization in the central Mediterranean region. The work built on the theoretical underpinnings of ANNs, CNNs and GNNs, prior to developing a GNN-based framework suited to the dataset and research aims.

Chapter 2 introduced the core concepts of ANNs, starting with artificial neurons, activation functions and the basic structure of an ANN by focusing on feedforward neural networks, specifically MLPs. The forward propagation procedure, describing how information travels through a feedforward neural network, was outlined. The chapter also delved into the UAT for feedforward neural networks, an important result that states that feedforward neural networks with sufficient neurons can approximate any continuous function. The training process was then explored by detailing the backpropagation algorithm, loss functions and optimizers.

Building on these concepts, Chapter 3 outlined CNNs, which adapt the principles of ANNs to exploit spatial hierarchies in data. Core concepts such as, convolution and pooling that aid the CNN to extract increasingly abstract features from structured inputs, such as waveform data, were introduced. Chapter 4 then generalized NN principles to graph-structured data. The main focus was given to the message passing mechanism between nodes, and the construction of edges based on similarity criteria. This enables the flexible modelling of relational dependencies that cannot be effectively captured by conventional ANNs and CNNs.

Together, these chapters provided the theoretical foundation required to develop the GNN framework capable of determining the seismic sources of earthquakes hap-

pening in the central Mediterranean region, by predicting their latitude, longitude, depth and magnitude. Training and validation were carried out on a dataset compiled from INGV earthquake catalogues and waveform recordings (see Section 1.1) covering the period 2013-2024, and geographic subset from 34° to 39° latitude and 11° to 16.5° longitude, a depth range of 0-30km, and a magnitude range of [3,5.5].

The application was carried out in Chapter 5. Initially, waveform features were extracted from the raw waveform metadata using CNN blocks, and combined with station locations and derived features, such as peak amplitude, RMS amplitude, spectral centroid and SNR (see Section 1.1). This combined input was then fed into the GNN models. Two architectures were considered, namely, an edgeless graph architecture (see Section 5.5) and dynamic edges GNN architecture (see Section 5.6) that constructed edges dynamically based on similarity criteria (see Section 4.4). Both architectures were trained via the AdamW optimizer with a weight decay parameter $1e-4$, minimizing Huber loss (see Section 2.4). Then, training performance was examined using loss curves, while model comparison was carried out with validation metrics MSE, MAE and NRMSE (see Section 2.7), as well as comparing actual versus predicted values, and examining model uncertainty.

Several experiments were conducted to better the generalization ability and training. These included hyperparameter tuning of batch size considering values 16, 32 and 64, learning rate taking values, $1e-5$, $1e-4$, $1e-3$ and $1e-2$, dropout rates of 0.2, 0.3, 0.4 and 0.5, and $k = \{3, 4\}$ in the dynamic edges GNN models, to identify the optimal configurations (see Sections 5.5.2 and 5.6.2). Further regularization strategies, such as early stopping (see Section 2.6) were also implemented. Additionally, cluster analysis was employed to restrict the models to a subset of the dataset (see Section 5.7), with the intention of reducing the variability within the training/validation sets by grouping similar events together, potentially enabling the models to capture more consistent patterns. This culminated in improved predictive accuracy. Ensemble modelling was also implemented in an attempt to further enhance predictive accuracy (see Section 5.8.2). However, this approach degraded performance, likely due to the similarity among candidate models, which diluted the strengths of the best individual models. After evaluating these experiments, the best performing edgeless graph model and dynamic edges GNN model were compared. Notably, the edgeless graph architecture outperformed the dynamic edges GNN architecture. When fit on independent test data having events from January 2025 to August 2025, in order to examiner final predictive ability, the edgeless graph model once again achieved superior architecture

performance (see Section 5.8).

It is worth noting that the immediate accuracy of the model is not perfect. Nonetheless, its demonstrated potential for rapid first-order characterization, contribution to early warning pipelines, identification of regional biases, and establishment of a baseline framework underscores its value as both a scientific tool and a stepping stone for advancing earthquake source characterization in the central Mediterranean region.

6.2 Limitations

Although the primary objective of this study was achieved, there are several limitations that should be acknowledged. Such limitations pertain mostly to computational resources, the dataset, and the lack of available literature.

Despite the availability of relatively strong computational resources (see Section 5.2), the size and complexity of the NN models imposed practical limitations. For instance, for the dynamic edges GNN architecture in Section 5.6, taking values of $k \geq 5$ was not feasible due to the substantial computational demand. Additionally, the maximum number of epochs that could be used was restricted to 300 due to computational constraints, which in turn may not have given the model enough time to learn. Moreover, these computational limitations also prevented the effective use of parallel processing during hyperparameter tuning, further increasing the total computation time.

The sample size and the nature of the dataset were significant limitations. The sizes of the training (548 events), validation (137 events) and test (26 events) sets were relatively small when compared with those present in the literature, which meant that the models did not always have sufficient data to learn the underlying patterns well, as well as restricted the models' capacity to generalize to unseen data. Moreover, the distribution of seismic stations and events was uneven, which negatively impacted prediction accuracy for earthquake location. The lack of large magnitude events further contributed degraded prediction accuracy, as the model's exposure to such instances were limited, making it challenging to output reliable predictions for larger earthquakes.

Lastly, the lack of established literature regarding seismic source characterization using GNNs posed a challenge. This is because there were little methodological precedents or benchmarks for comparison.

6.3 Further Research and Recommendations

In view of the findings, the following recommendations for future research are proposed. Firstly, the relatively small sample size could be addressed by expanding the dataset as new seismic events are recorded. Additionally, synthetic sampling methods, like some form of data augmentation or over-sampling strategies, could be employed to create a more balanced dataset.

The improvement in predictive performance warrants further investigation. Future research could explore more advanced ensemble modelling approaches, for example by applying weighting mechanisms or incorporating a greater diversity of model architectures, with the aim of combining complementary strengths. Additionally, further optimization could be pursued through a broader exploration of model configurations, such as varying the number of layers, adjusting weight decay parameters, and evaluating more extensive hyperparameter combinations.

Finally, future work could focus on establishing stronger quantitative baselines by implementing existing earthquake source characterization methods from the literature and evaluating their performance on the dataset compiled in this study. Such comparisons would enable a more direct and objective assessment of the performance gains attributable to the proposed approach.

Appendix A

Illustrative Example of Spectral Features Derived from Waveforms

In this appendix an numerical example illustrating the computation of the spectral features derived from the raw waveforms is provided. The material presented here is meant solely to provide an example and it should be noted that this notation is independent of, and should not be confused with, the notation used throughout the main body of the dissertation.

Consider a simple synthetic waveform trace comprising 16 samples, sampled at 4 Hz (total duration of 4 seconds). The values of the trace are:

$$x_n = [0, 2, 3, 4, 6, 7, 5, 3, 2, 1, 0, 0, 0, 0, 0, 0],$$

where $n = 1, ..., N$, $N = 16$. For the purpose of this example, the noise window is taken to be 0-1s and the signal window 1-3s. The peak amplitude, RMS amplitude, spectral centroid and SNR can be computed as follows.

Peak Amplitude

The peak amplitude is defined as the *maximum absolute value of the trace*. In this example, the largest value is 7, so the peak amplitude is 7.

RMS Amplitude

The RMS Amplitude is computed by squaring each sample, averaging these squared values and taking the square root. Thus,

$$\text{RMS Amplitude} = \sqrt{\frac{2^2 + 3^2 + 4^2 + 6^2 + 7^2 + 5^2 + 3^2 + 2^2 + 1^2}{16}} \approx 3.09.$$

Spectral Centroid

The Spectral Centroid is a frequency-domain measure obtained by computing the real Fast Fourier Transform (FFT) of the signal and taking a weighted average of the frequency bins, where each weight is the magnitude of the corresponding FFT coefficient. Thus, the spectral centroid of a trace can be defined as:

$$f_c = \frac{\sum_{k=0}^{N/2} f_k |X(k)|}{\sum_{k=0}^{N/2} |X(k)|}, \tag{A.1}$$

where N is the number of time-domain samples in the signal and k is the index of the FFT frequency bin. Furthermore, f_k is the frequency associated with the k^{th} FFT bin, given by $f_k = \frac{k}{N} f_s$, for a sampling frequency f_s, and $X(k)$ is the Discrete Fourier Transform (DFT) of the signal.

In our case, $N = 16$, $f_s = 4$ Hz and taking non-negative components of x_n, we have $k = 0, ..., 8$. Thus,

$$f_k = \frac{k}{N} f_s = 0.25k.$$

In order to compute $X(k)$, we use the formula:

$$X(k) = \sum_{n=0}^{N-1} x_n e^{\frac{-i2\pi kn}{N}}. \tag{A.2}$$

Now, for $k = 0$, the exponential becomes 1, and so:

$$X(0) = \sum_{n=0}^{N-1} x_n = 0 + 2 + 3 + \cdots + 0 = 33,$$

thus, $|X(0)| = |33| = 33$. Then for example, for $k = 1$, we have:

$$X(1) = \sum_{n=0}^{N-1} x_n e^{\frac{-i\pi n}{8}}. \tag{A.3}$$

Letting $\theta_n = \frac{\pi n}{8}$, we can use the identity $e^{-i\theta} = \cos\theta - i\sin\theta$, thus, Equation (A.3) can be re-written as:

$$\begin{aligned} X(1) &= \sum x_n \cos(\theta_n) - i \sum x_n \sin(\theta_n) \\ &= 0\cos(0) - 0i\sin(0) + 2\cos\left(\frac{\pi}{8}\right) - 2i\sin\left(\frac{\pi}{8}\right) + \cdots + \cos\left(\frac{9\pi}{8}\right) - i\sin\left(\frac{9\pi}{8}\right) \\ &= -6.41 - 23.3503i. \end{aligned}$$

So, $|X(1)| = \sqrt{(-6.41)^2 + (-23.3503)^2} \approx 24.2141$. Therefore, using the definition of f_k and Equation (A.2), the components required to compute the spectral centroid f_c

using Equation (A.1) can be computed and are summarized in Table A.1.

k	f_k (Hz)	$\lvert X(k)\rvert$	$f_k\lvert X(k)\rvert$ (Hz)
0	0	33	0
1	0.25	24.2141	6.0535
2	0.5	8.5889	4.2945
3	0.75	1.6591	1.2443
4	1	3	3
5	1.25	2.5977	3.2471
6	1.5	0.48	0.7201
7	1.75	0.42	0.7250
8	2	1	2

Table A.1: Components required to compute spectral centroid f_c.

Finally, using Equation A.1, substituting the components in Table A.1, the spectral centroid is given by:

$$f_c \approx \frac{74.9598}{21.2945} \approx 0.2841 \text{ Hz}.$$

SNR

The SNR is calculated using the formula:

$$SNR = \frac{RMS\ Amplitude_{signal}}{RMS\ Amplitude_{noise}}. \tag{A.4}$$

Now, noise samples correspond to the first four values $[0, 2, 3, 4]$, and signal samples correspond to the next eight $[6, 7, 5, 3, 2, 0, 0]$. Therefore,

$$RMS\ Amplitude_{noise} = \sqrt{\frac{2^2 + 3^2 + 4^2}{4}} \approx 2.69,$$

and

$$RMS\ Amplitude_{signal} = \sqrt{\frac{6^2 + 7^2 + 5^2 + 3^2 + 2^2 + 1^2}{8}} \approx 3.94.$$

Thus,

$$SNR \approx \frac{3.94}{2.69} \approx 1.46.$$

Appendix B

Functional Analysis Theory required for Universal Approximation Theorem

In this appendix several fundamental definitions and theorem statements from functional analysis that are necessary for a complete understanding of the proof of Theorem 1 are provided. The material presented here is meant solely to provide the mathematical background required for the proof and it should be noted that this notation is independent of, and should not be confused with, the notation used throughout the main body of the dissertation.

Definition 2 Linear Space

A linear space *(or* vector space*)* X *over a scalar field* $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$ *is a set of elements, called* vectors*, together with two operations:*

- ***Vector addition:*** *a map* $+ : X \times X \to X$*,*
- ***Scalar multiplication:*** *a map* $\cdot : \mathbb{K} \times X \to X$*,*

such that the following properties hold:

(a) ***Additive group structure.***

$(X, +)$ *is a commutative group:*

- $x + y = y + x$*,* $\forall x, y \in X$ *(commutativity),*
- $x + (y + z) = (x + y) + z$*,* $\forall x, y, z \in X$ *(associativity),*
- $\exists\ 0 \in X$ *such that* $x + 0 = x$*,* $\forall x \in X$ *(identity element),*
- $\forall x \in X$*,* $\exists\ -x \in X$ *such that* $x + (-x) = 0$ *(inverse element).*

(b) ***Scalar multiplication axioms.***

$\forall x, y \in X$ *and* $\forall \alpha, \beta \in \mathbb{K}$*:*

- $\alpha(x+y) = \alpha x + \alpha y$ *(distributivity over vector addition),*
- $(\alpha+\beta)x = \alpha x + \beta x$ *(distributivity over scalar addition),*
- $(\alpha\beta)x = \alpha(\beta x)$ *(compatibility with field multiplication),*
- $1 \cdot x = x$ *(multiplicative identity).*

Definition 3 Linear Subspace

A linear subspace of X is a subset $U \subseteq X$ that is closed under vector addition and scalar multiplication, i.e. $\forall x, y \in U$ and $\alpha \in \mathbb{K}$,

$$x + y \in U \quad \textit{and} \quad \alpha x \in U.$$

Definition 4 Norm

A norm on a linear space X is a function $||\cdot|| : X \to \mathbb{K}$ with the following properties:

(a) $||x|| \geq 0, \ \forall x \in X$ *(non-negativity),*

(b) $||\alpha x|| = |\alpha| ||x||, \ \forall x \in X, \alpha \in \mathbb{K}$ *(homogeneity),*

(c) $||x+y|| \leq ||x|| + ||y||, \ \forall x, y \in X$ *(triangle inequality),*

(d) $||x|| = 0 \implies x = 0.$

Definition 5 Normed Linear Space

A normed linear space $(X, ||\cdot||)$ is a linear space X equipped with a norm $||\cdot||$.

Definition 6 Supremum Norm

Let S be a set, $(X, ||\cdot||)$ be a normed space and $\mathcal{B}$ be the set of bounded mappings $S \to X$. For $f \in \mathcal{B}$, the supremum norm of f on S is defined as:

$$||f||_\infty = \sup\{||f(x)|| : x \in S\}.$$

Definition 7 Supremum Norm on Continuous Closed Interval of Real Valued Functions

Let $I = [a...b]$ be a closed interval, $C(I)$ be the space of real-valued function continuous on I and $f \in C(I)$. Suppose $|\cdot|$ denotes the absolute value and sup *denotes the supremum of real-valued functions. The supremum norm over $C(I)$ is defined as:*

$$||f||_\infty = \sup_{x \in I} |f(x)|.$$

Definition 8 Linear Map

Let $\phi : X \to Y$ be a map between linear spaces X and Y, ϕ is said to be linear if $\forall x, y \in X, \alpha \in \mathbb{K}$:

$$\phi(x + \alpha y) = \phi x + \alpha \phi y.$$

Definition 9 Linear Functional

Let X be a normed space over a field $\mathbb{K}$, if $\phi : X \to \mathbb{K}$ is a linear map, ϕ is said to be a linear functional.

Definition 10 Proper Subset

A set S is a proper subset of a set T, $S \subsetneq T$, if S is a subset of T and $S \neq T$.

Theorem 3 Hahn-Banach Theorem for Normed Linear Spaces (Banach, 1929a,b; Hahn, 1927)

Let X be a real or complex normed linear space, let $M \subseteq X$ be a linear subspace, and let $\phi \in M^*$, where M^* is the dual space of M, be a bounded linear functional on M. Then $\exists$ a linear functional $\tilde{\phi}$ that extends ϕ (i.e. $\tilde{\phi} \restriction M = \phi$) and satisfies $||\tilde{\phi}||_{X^*} = ||\phi||_{M^*}$.

Given a topological space X, define $C(X)$ as the complex vector space of continuous functions from X into M under pointwise addition and pointwise scaling. Denote the collection of functions $f \in C(X)$ whose support $supp(f) := cl_X\{x \in X : f(x) \neq 0\}$ is compact by $C_c(X)$. Let μ be a locally finite Borel measure on X. The statement for the Riesz-Markov-Kakutani Representation Theorem is as follows:

Theorem 4 Riesz-Markov-Kakutani Representation Theorem (Kakutani, 1941; Markov, 1938; Riesz, 1909)

Let X be a locally compact Hausdorff space. If ϕ is a positive linear functional on $C_c(X)$, then $\exists$ a unique Radon measure μ on X such that:

$$\phi(f) = \int f d\mu, \ \forall f \in C_c(X).$$

Appendix C

Further Regularization Techniques

In this section, we present the theoretical background on ANN regularization techniques, namely, L_1 *and* L_2 *regularization* and *batch normalization*, referenced in Chapter 2. The aim is to clarify these concepts that are mentioned throughout the dissertation but are not employed directly in the Chapter 5 application. For coherence, the notation introduced here is aligned with the conventions used in the main body of the dissertation.

C.1 L_1 and L_2 Regularization

L_1 and L_2 regularization work by adding a penalty function to either of the loss functions previously introduced in Section 2.4 to prevent over-fitting. This is done so that a bias towards simpler models is introduced, discouraging complex relationships and unnecessarily large weights. Ng (2004) played a key role in promoting both L_1 and L_2 regularization in statistical learning context, highlighting when to use which method.

L_1 regularization is typically made use of in instances where the model might have redundant features. The penalty added to the loss function is proportional to the absolute value of the weights. This results in certain weights taking on a zero value, removing any irrelevant features from the model. Consider two consecutive layers l and $l+1$, the L_1 penalty is given by:

$$L_1 = \lambda \sum_{l=1}^{L} \sum_{i,i'} |w_{i,i'}^{(l,l+1)}|. \tag{C.1}$$

Here, λ is the regularization parameter, L is the final layer in the network, and $w_{i,i'}^{(l,l+1)}$ is the weight joining the i^{th} neuron in layer l to the i'^{th} neuron in layer $l+1$. On the other hand, in L_2 regularization, the penalty introduced to the loss function is proportional

to the square of the weights, as can be seen from Equation (C.2). This results in greater penalties being given to weights which are larger. Thus, L_2 regularization encourages the model to use smaller weights, resulting in a better distribution of weights:

$$L_2 = \lambda \sum_{l=1}^{L} \sum_{i,i'} (w_{i,i'}^{(l,l+1)})^2 \tag{C.2}$$

where λ, L,and $w_{i,i'}^{(l,l+1)}$ are defined as before. In applications, L_1 and L_2 regularization are often used in conjunction. This is termed as elastic net regularization (Zou & Hastie, 2005).

C.2 Batch Normalization

Adjusting NN parameters during training can alter the statistical distributions characteristics of activations. These characteristics include the mean, variance, and range of the outputs from one layer, which serve as inputs to the next. These shifts can affect both the stability and efficiency of training. This is known as *internal covariate shift*. In order to tackle this issue, Ioffe & Szegedy (2015) proposed *batch normalization*, a regularization technique that reduces internal covariate shift by normalizing the inputs to every layer within a mini-batch, in turn mitigating the vanishing/exploding gradient problem. A mini-batch is a subset of the training data which is made use of during one iteration of a NN training process. The mini-batch's size is a hyperparameter which depends on the data. Larger batch sizes can enhance training speed when parallel processing hardware, such as GPUs, is used effectively. In contrast, smaller batch sizes introduce more noise in gradient estimates which can in turn improve generalization to unseen data and lead to better convergence in some cases (Goodfellow et al., 2016).

Batch normalization is carried out using the following steps. Suppose batch normalization is applied between two layers l and $l+1$ with total number of neurons n_l and n_{l+1}, respectively. For a given mini-batch of size m, let:

$$\{\boldsymbol{z}_1^l, ..., \boldsymbol{z}_m^l\}$$

be the activations (or inputs) to layer $(l+1)$, where each $\boldsymbol{z}_{\tilde{k}}^l$, $\tilde{k} = 1, ..., m$, is an n_l-vector. The mini-batch mean and variance is computed for each $i \in \{1, ..., n_l\}$,

$$\text{mean} = \bar{z}_{Bi}^l = \frac{1}{m} \sum_{\tilde{k}=1}^{m} z_{\tilde{k}i}^l \tag{C.3}$$

$$\text{variance} = (s_{Bi}^l)^2 = \frac{1}{m}\sum_{\tilde{k}=1}^{m}(z_{\tilde{k}i}^l - \bar{z}_{Bi}^l)^2 \tag{C.4}$$

then, each activation component is normalized using:

$$\hat{z}_{\tilde{k}i}^l = \frac{z_{\tilde{k}i}^l - \bar{z}_{Bi}^l}{\sqrt{(s_{Bi}^l)^2 + \tilde{\varepsilon}}} \tag{C.5}$$

where $\tilde{\varepsilon}$ is a negligible constant added for numerical stability and to avoid dividing by zero. By default, $\tilde{\varepsilon}$ is typically set automatically by the software. For instance, in the python module `PyTorch` the default is $\tilde{\varepsilon} = 1\times10^{-5}$ (PyTorch Contributors, 2024), while in python module `TensorFlow` it is 1×10^{-3} (Keras Team, 2025). Although most users keep these defaults, it is also possible to override them if particular training scenarios require a different value. Lastly, the normalized values are re-scaled and shifted as follows:

$$\xi_{\tilde{k}i} = \kappa_i \hat{z}_{\tilde{k}i} + \tau_i. \tag{C.6}$$

Here κ_i and τ_i are the scaling and shifting parameters for the i^{th} neuron and $\xi_{\tilde{k}i}$ is the batch normalization output. Despite the scaling and shifting parameters needing learning during the training process, computation time is not significantly increased. So, in practice, batch normalization is typically applied to the outputs of a layer prior to implementing the activation function.

Appendix D

The Weisfeiler-Lehman Algorithm and the Expressiveness of GNNs

The Weisfeiler-Leman (WL) algorithm (Weisfeiler & Leman, 1968), also referred to as the *state refinement* algorithm, is a method developed to test for graph *isomorphism*. With regards to this dissertation, its importance lies in its close connection to the expressive power of GNNs which was discussed in Section 4.5. Prior to presenting the WL algorithm, it is useful to define the concept of graph isomorphism:

Definition 11 Graph Isomorphism
Two graphs $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ *and* $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ *are said to be isomorphic if* $\exists$ *a bijective function* $\psi : \mathcal{V}_1 \to \mathcal{V}_2$ *s.t.* $(u, v) \in \mathcal{E}_1 \iff (\psi(u), \psi(v)) \in \mathcal{E}_2$.

Intuitively, two graphs are isomorphic if they have the same structure, but the nodes might be presented in different order. In the WL algorithm, each node u at iteration t is assigned a state $s_u^{(t)}$, which is a discrete label that encodes the structural information available to the node at that iteration. The WL algorithm iteratively refines the node states by combining each node's current state with the multi-set of states of its neighbours. At the t^{th} iteration, the algorithm proceeds as follows:

1. Each node u has associated with it a state $s_u^{(t)}$. Initially, at $t = 0$, all nodes may be given the same state.
2. The state of each node is updated by encoding the pair consisting of its current state and the multi-set of its neighbours' states.
3. These encoded values are then compressed into new discrete states, producing $s_u^{(t+1)}$.

This process is repeated for successive iterations, gradually refining the node states. The pseudocode for the WL algorithm is presented in Algorithm 1.

Algorithm 1 State Refinement

$t \leftarrow 0$
for all $u \in \mathcal{V}$ **do**
  $s_u^{(0)} \leftarrow 0$
end for
while True **do**
  for all $u \in \mathcal{V}$ **do**
    $s_u^{(t+1)} = \text{RELABEL}\Big(s_u^{(t)}, \{\{ s_v^{(t)} \mid v \in \mathcal{N}(v) \}\}\Big)$
  end for
  $t \leftarrow t + 1$
end while

The RELABEL function in Algorithm 1 is simply a hash function that assigns a different state to every possible configuration in the node neighbourhood. More formally, the RELABEL function is an injection $\text{RELABEL} : \mathbb{R} \times \mathcal{M} \to \mathbb{R}$, where $\mathcal{M}$ represents all the possible multi-sets of $\mathbb{R}$. Intuitively, this implies that if two nodes u and v have the same state at iteration t, then they will receive a different state in the subsequent iteration, if and only if, there exists a state s such that u and v have different amount of neighbours that are in state s at iteration t. After the first round of the algorithm, two nodes have the same state if and only if they have the same degree. It is worth mentioning, that in the pseudocode there is no termination condition. Nevertheless, in practice, one is only interested in further refinement steps as long as progress is made, i.e., as long as the algorithm distinguishes additional/nodes. Formally, the termination condition can be defined as in Definition 12.

Definition 12 WL Termination

The refinement state is said to be finished at iteration t, if $\nexists$ two nodes u and v s.t. $s_u^{(t)} = s_v^{(t)}$, but $s_u^{(t+1)} \neq s_v^{(t+1)}$.

It is worth noting the WL algorithm terminates in at most N iterations, where N is the number of nodes in the graph (Babai & Kucera, 1979). Now given two graphs, $\mathcal{G}_1$ and $\mathcal{G}_2$, the WL algorithm can be applied separately to both in order to determine whether they are isomorphic. After every refinements step, the multi-sets M_1 and M_2 of node states corresponding to the graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ are compared:

- if the multi-sets differ at some iteration, then the graphs are certainly not isomorphic,
- if the multi-sets are identical, the algorithm cannot conclude; the graphs may still be non-isomorphic.

Thus, WL provides a one-sided isomorphism test. Checking quality of multi-sets can

be done, for example, by sorting the states and comparing element by element. For the full power of the algorithm, it is necessary to compare the actual states in multi-sets M_1 and M_2, and not solely the number of occurrences of each state. For instance, let $\mathcal{G}_1$ be a clique[1] of 4 nodes, and $\mathcal{G}_2$ be a cycle[2] of length 4. An example of a 4-clique and a 4-cycle can be seen in Figure D.1. Both graphs are regular, meaning every node has the same degree. Thus, the refinement procedure will terminate after one iteration in both cases, and both graphs will have 4 nodes of identical state. Nevertheless, the resulting states differ between the two graphs. This is because in $\mathcal{G}_1$ each node has degree 3, whereas in $\mathcal{G}_2$ each node is of degree 2. As a result, $M_1 \neq M_2$ and the WL algorithm deems these two graphs different.

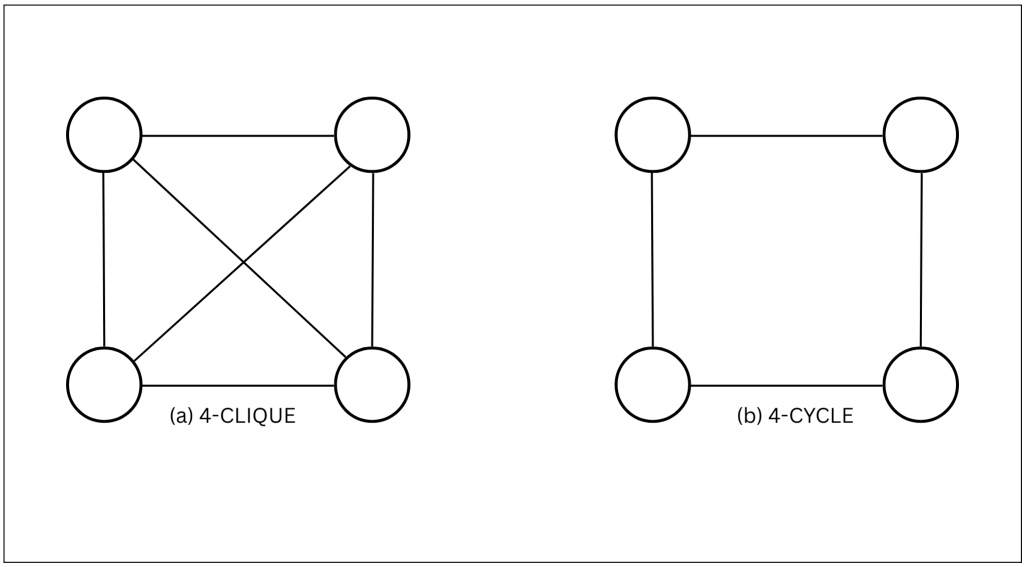

Figure D.1: An example of a (a) 4-CLIQUE and a (b) 4-CYCLE.

More generally, if both graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ are k-regular, i.e., every node has degree exactly k, then the algorithm cannot distinguish them, since it already terminates in the first iteration (Morris et al., 2023). However, it is worth noting that the majority of graphs can be distinguished by the WL algorithm.

The WL algorithm is closely related to the expressive power of GNNs. This is because if two nodes u and v are initialized with the same features, i.e. $\boldsymbol{h}_u^{(0)} = \boldsymbol{h}_v^{(0)}$, then after message passing, their embeddings evolve analogously to WL state refinement. As a result, the expressivity of standard message passing GNNs is bounded above by the WL algorithm. This suggests that if the WL algorithm cannot distinguish between two graphs, then any such GNN will also generate the same output.

At the same time, it can be shown that GNNs are as powerful as the WL algo-

[1] A **clique** is a complete graph when every pair of distinct nodes is connected.
[2] A **cycle** is a graph where the nodes form a single closed loop.

rithm. This means that for every function that can be expressed in the message passing model of WL refinement, there exists a GNN that can also express it. This equivalence provides the basis for theoretical results on the universality of GNNs. Loosely speaking, it demonstrates that whenever an algorithm can be formulated in terms of local message passing, then a suitably designed GNN can in theory represent it. Whether such functions can be efficiently learned in practice, however, is a different matter. In fact, the WL algorithm is a hierarchy of increasing in power heuristics. What was discussed in this section is simply the 1-WL algorithm, which refines states based on individual neighbourhoods. More expressive variants, known as k-WL algorithms, refine states over k-tuples of nodes. In actual fact, Morris et al. (2019) demonstrated that k-GNNs can be expressive as much as the k-WL algorithm. Nevertheless, the k-WL test and k-GNNs are computationally intensive since as k increases, the time complexity of the message passing explores combinatorially (Morris et al., 2019).

Appendix E

Links Containing Additional Material

The relevant Python codes can be accessed through Github under the name SOR-5200-MSc-in-Statistics-Dissertation-Python-Codes by clicking here. The loss plots and the predictions versus actual plots can be accessed through Google Drive by clicking here.

Appendix F

Schematics of Architecture Components

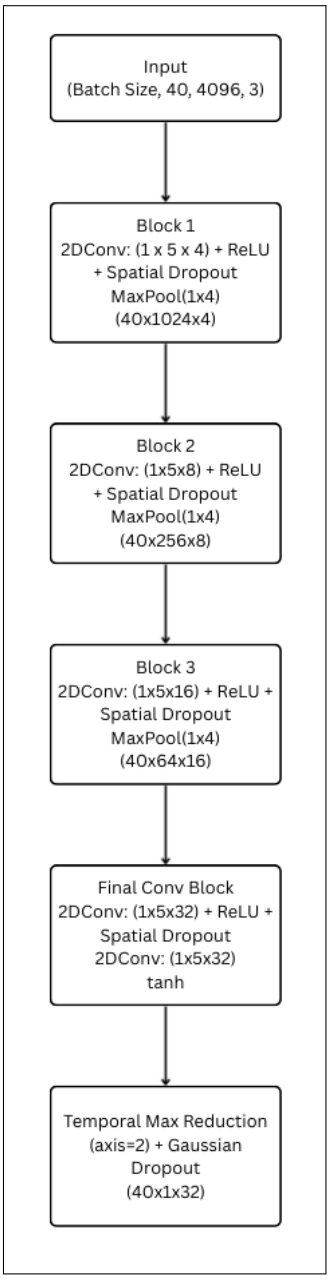

Figure F.1: Schematic of CNN-Based Waveform Encoder.

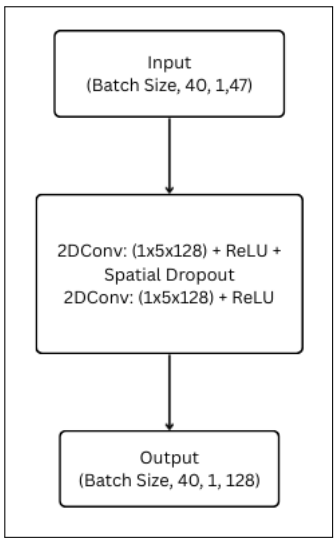

Figure F.2: Schematic of Feature Fusion Block of Edgeless Graph Architecture.

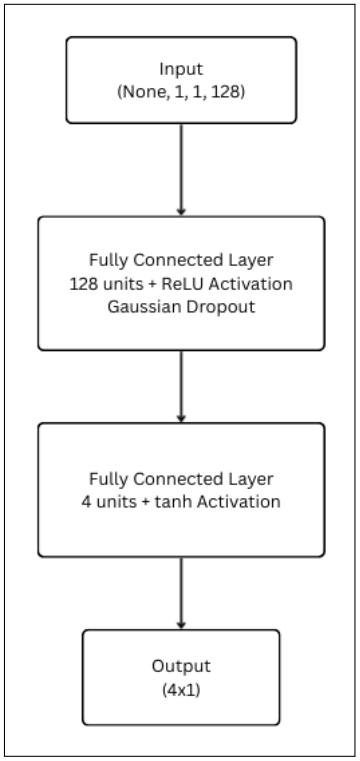

Figure F.3: Schematic of MLP that generates the final predicted output.

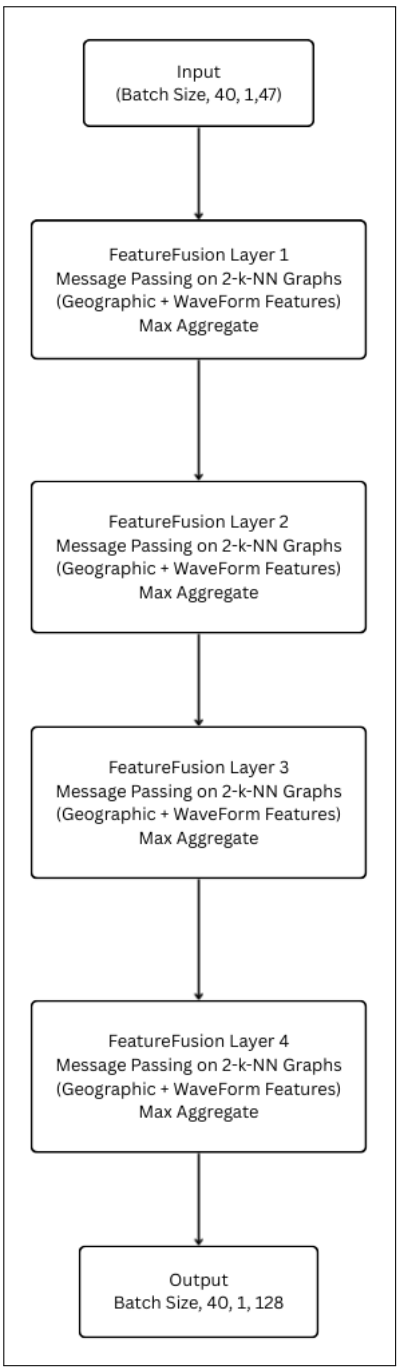

Figure F.4: Schematic of Feature Fusion Block of Dynamic Edges GNN Architecture.

Appendix G

Summary of Performance Metrics for Hyperparameter Tuning using Edgeless Graph Model on Full Dataset

Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
16	0.00001	0.2	1.14	2.19	1.02
16	0.00001	0.3	1.11	2.03	0.99
16	0.00001	0.4	1.19	2.75	1.15
16	0.00001	0.5	1.38	3.74	1.34
16	0.0001	0.2	0.84	1.26	0.78
16	0.0001	0.3	1.08	2.02	0.98
16	0.0001	0.4	1.14	2.22	1.03
16	0.0001	0.5	1.15	2.11	1.01
16	0.001	0.2	0.81	1.23	0.77
16	0.001	0.3	0.80	1.23	0.77
16	0.001	0.4	0.88	1.43	0.83
16	0.001	0.5	0.93	1.49	0.85
16	0.01	0.2	1.14	2.32	1.05
16	0.01	0.3	1.14	2.37	1.07
16	0.01	0.4	1.15	2.31	1.05
16	0.01	0.5	1.14	2.35	1.06
32	0.00001	0.2	1.14	2.30	1.05
32	0.00001	0.3	1.21	2.74	1.15
32	0.00001	0.4	1.42	3.88	1.36
32	0.00001	0.5	1.58	4.57	1.48
32	0.0001	0.2	1.04	1.84	0.94
32	0.0001	0.3	1.11	2.14	1.01
32	0.0001	0.4	1.14	2.15	1.02
32	0.0001	0.5	1.17	2.11	1.01
32	0.001	0.2	0.76	1.13	0.74
32	0.001	0.3	0.84	1.31	0.79
32	0.001	0.4	0.87	1.42	0.83
32	0.001	0.5	1.13	2.32	1.05
32	0.01	0.2	0.92	1.62	0.88
32	0.01	0.3	0.90	1.49	0.85
32	0.01	0.4	1.14	2.35	1.06
32	0.01	0.5	1.14	2.35	1.06

Table G.1: Validation performance metrics for latitude using the Edgeless Graph Architecture (part 1).

Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
64	0.00001	0.2	1.13	2.27	1.04
64	0.00001	0.3	1.31	3.35	1.27
64	0.00001	0.4	1.42	3.95	1.38
64	0.00001	0.5	1.20	2.06	0.99
64	0.0001	0.2	1.13	2.20	1.03
64	0.0001	0.3	1.11	2.05	0.99
64	0.0001	0.4	1.13	2.06	0.99
64	0.0001	0.5	1.53	4.36	1.45
64	0.001	0.2	0.79	1.15	0.74
64	0.001	0.3	0.82	1.23	0.77
64	0.001	0.4	0.83	1.22	0.76
64	0.001	0.5	1.14	2.32	1.05
64	0.01	0.2	0.78	1.24	0.77
64	0.01	0.3	0.88	1.44	0.83
64	0.01	0.4	1.13	2.35	1.06
64	0.01	0.5	1.14	2.35	1.06

Table G.2: Validation performance metrics for latitude using the Edgeless Graph Architecture (part 2).

Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
16	0.00001	0.2	0.70	0.85	1.03
16	0.00001	0.3	0.68	0.82	1.01
16	0.00001	0.4	0.92	1.33	1.28
16	0.00001	0.5	1.57	3.23	2.00
16	0.0001	0.2	0.65	0.81	1.00
16	0.0001	0.3	0.67	0.81	1.00
16	0.0001	0.4	0.69	0.81	1.00
16	0.0001	0.5	0.69	0.82	1.01
16	0.001	0.2	0.65	0.80	1.00
16	0.001	0.3	0.65	0.79	0.99
16	0.001	0.4	0.66	0.82	1.01
16	0.001	0.5	0.65	0.79	0.99
16	0.01	0.2	0.69	0.81	1.00
16	0.01	0.3	0.69	0.81	1.00
16	0.01	0.4	0.69	0.81	1.00
16	0.01	0.5	0.69	0.81	1.00
32	0.00001	0.2	0.77	1.05	1.14
32	0.00001	0.3	0.80	1.09	1.16
32	0.00001	0.4	1.38	2.60	1.80
32	0.00001	0.5	1.44	2.86	1.88
32	0.0001	0.2	0.66	0.82	1.01
32	0.0001	0.3	0.68	0.81	1.00
32	0.0001	0.4	0.69	0.81	1.00
32	0.0001	0.5	0.69	0.83	1.01
32	0.001	0.2	0.64	0.80	1.00
32	0.001	0.3	0.66	0.79	0.99
32	0.001	0.4	0.65	0.79	0.99
32	0.001	0.5	0.68	0.80	1.00
32	0.01	0.2	0.67	0.81	1.00
32	0.01	0.3	0.66	0.80	1.00
32	0.01	0.4	0.69	0.81	1.00
32	0.01	0.5	0.69	0.81	1.00

Table G.3: Validation performance metrics for longitude using the Edgeless Graph Architecture (part 1).

Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
64	0.00001	0.2	0.77	1.04	1.14
64	0.00001	0.3	1.28	2.30	1.69
64	0.00001	0.4	1.54	3.18	1.99
64	0.00001	0.5	1.25	2.27	1.68
64	0.0001	0.2	0.70	0.81	1.00
64	0.0001	0.3	0.68	0.83	1.01
64	0.0001	0.4	0.71	0.86	1.03
64	0.0001	0.5	1.52	3.10	1.96
64	0.001	0.2	0.65	0.84	1.02
64	0.001	0.3	0.65	0.82	1.01
64	0.001	0.4	0.64	0.80	1.00
64	0.001	0.5	0.69	0.81	1.00
64	0.01	0.2	0.65	0.79	0.99
64	0.01	0.3	0.66	0.81	1.00
64	0.01	0.4	0.67	0.80	1.00
64	0.01	0.5	0.69	0.81	1.00

Table G.4: Validation performance metrics for longitude using the Edgeless Graph Architecture (part 2).

Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
16	0.00001	0.2	6.41	54.27	1.01
16	0.00001	0.3	6.40	54.67	1.01
16	0.00001	0.4	6.55	57.87	1.04
16	0.00001	0.5	7.01	65.63	1.11
16	0.0001	0.2	6.29	55.32	1.02
16	0.0001	0.3	6.35	54.01	1.00
16	0.0001	0.4	6.39	54.18	1.00
16	0.0001	0.5	6.38	53.76	1.00
16	0.001	0.2	6.58	64.58	1.10
16	0.001	0.3	6.34	55.86	1.02
16	0.001	0.4	6.22	54.06	1.00
16	0.001	0.5	6.19	53.48	1.00
16	0.01	0.2	6.51	56.17	1.02
16	0.01	0.3	6.49	55.90	1.02
16	0.01	0.4	6.52	56.62	1.03
16	0.01	0.5	6.50	56.14	1.02
32	0.00001	0.2	6.40	54.56	1.01
32	0.00001	0.3	6.25	52.79	0.99
32	0.00001	0.4	6.85	64.07	1.09
32	0.00001	0.5	6.53	59.53	1.05
32	0.0001	0.2	6.38	55.72	1.02
32	0.0001	0.3	6.32	53.51	1.00
32	0.0001	0.4	6.39	54.02	1.00
32	0.0001	0.5	6.42	54.08	1.00
32	0.001	0.2	6.35	59.90	1.06
32	0.001	0.3	6.36	56.97	1.03
32	0.001	0.4	6.26	54.68	1.01
32	0.001	0.5	6.25	53.52	1.00
32	0.01	0.2	6.43	54.86	1.01
32	0.01	0.3	6.34	53.31	1.00
32	0.01	0.4	6.52	56.57	1.03
32	0.01	0.5	6.50	55.96	1.02

Table G.5: Validation performance metrics for depth using the Edgeless Graph Architecture (part 1).

Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
64	0.00001	0.2	6.45	58.52	1.04
64	0.00001	0.3	6.71	61.65	1.07
64	0.00001	0.4	7.62	82.60	1.24
64	0.00001	0.5	10.41	152.28	1.68
64	0.0001	0.2	6.38	54.33	1.01
64	0.0001	0.3	6.34	54.80	1.01
64	0.0001	0.4	6.38	53.96	1.00
64	0.0001	0.5	6.62	58.61	1.05
64	0.001	0.2	6.36	57.30	1.03
64	0.001	0.3	6.43	59.20	1.05
64	0.001	0.4	6.36	56.73	1.03
64	0.001	0.5	6.34	54.07	1.00
64	0.01	0.2	6.36	57.91	1.04
64	0.01	0.3	6.41	55.36	1.02
64	0.01	0.4	6.36	53.31	1.00
64	0.01	0.5	6.47	55.41	1.02

Table G.6: Validation performance metrics for depth using the Edgeless Graph Architecture (part 2).

Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
16	0.00001	0.2	0.30	0.18	1.20
16	0.00001	0.3	0.28	0.14	1.06
16	0.00001	0.4	0.35	0.24	1.38
16	0.00001	0.5	0.36	0.25	1.41
16	0.0001	0.2	0.28	0.13	1.02
16	0.0001	0.3	0.28	0.13	1.02
16	0.0001	0.4	0.28	0.14	1.06
16	0.0001	0.5	0.27	0.13	1.02
16	0.001	0.2	0.27	0.12	0.98
16	0.001	0.3	0.27	0.12	0.98
16	0.001	0.4	0.27	0.13	1.02
16	0.001	0.5	0.27	0.12	0.98
16	0.01	0.2	0.28	0.13	1.02
16	0.01	0.3	0.28	0.13	1.02
16	0.01	0.4	0.28	0.13	1.02
16	0.01	0.5	0.28	0.13	1.02
32	0.00001	0.2	0.34	0.23	1.35
32	0.00001	0.3	0.33	0.22	1.32
32	0.00001	0.4	0.36	0.24	1.38
32	0.00001	0.5	0.35	0.24	1.38
32	0.0001	0.2	0.28	0.14	1.06
32	0.0001	0.3	0.28	0.14	1.06
32	0.0001	0.4	0.28	0.14	1.06
32	0.0001	0.5	0.28	0.14	1.06
32	0.001	0.2	0.27	0.11	0.94
32	0.001	0.3	0.27	0.12	0.98
32	0.001	0.4	0.27	0.12	0.98
32	0.001	0.5	0.28	0.13	1.02
32	0.01	0.2	0.27	0.13	1.02
32	0.01	0.3	0.27	0.13	1.02
32	0.01	0.4	0.28	0.13	1.02
32	0.01	0.5	0.28	0.13	1.02

Table G.7: Validation performance metrics for magnitude using the Edgeless Graph Architecture (part 1).

Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
64	0.00001	0.2	0.33	0.21	1.29
64	0.00001	0.3	0.35	0.24	1.38
64	0.00001	0.4	0.35	0.23	1.35
64	0.00001	0.5	0.33	0.22	1.32
64	0.0001	0.2	0.28	0.14	1.06
64	0.0001	0.3	0.28	0.14	1.06
64	0.0001	0.4	0.28	0.14	1.06
64	0.0001	0.5	0.36	0.25	1.41
64	0.001	0.2	0.28	0.12	0.98
64	0.001	0.3	0.28	0.13	1.02
64	0.001	0.4	0.28	0.12	0.98
64	0.001	0.5	0.28	0.13	1.02
64	0.01	0.2	0.27	0.12	0.98
64	0.01	0.3	0.28	0.13	1.02
64	0.01	0.4	0.28	0.13	1.02
64	0.01	0.5	0.28	0.13	1.02

Table G.8: Validation performance metrics for magnitude using the Edgeless Graph Architecture (part 2).

Batch Size	LR	Dropout Prob	NRMSE
16	0.00001	0.2	1.06
16	0.00001	0.3	1.02
16	0.00001	0.4	1.21
16	0.00001	0.5	1.46
16	0.0001	0.2	0.95
16	0.0001	0.3	1.00
16	0.0001	0.4	1.02
16	0.0001	0.5	1.01
16	0.001	0.2	0.96
16	0.001	0.3	0.94
16	0.001	0.4	0.96
16	0.001	0.5	0.95
16	0.01	0.2	1.02
16	0.01	0.3	1.03
16	0.01	0.4	1.03
16	0.01	0.5	1.03
32	0.00001	0.2	1.14
32	0.00001	0.3	1.16
32	0.00001	0.4	1.41
32	0.00001	0.5	1.45
32	0.0001	0.2	1.01
32	0.0001	0.3	1.02
32	0.0001	0.4	1.02
32	0.0001	0.5	1.02
32	0.001	0.2	0.93
32	0.001	0.3	0.95
32	0.001	0.4	0.95
32	0.001	0.5	1.02
32	0.01	0.2	0.98
32	0.01	0.3	0.96
32	0.01	0.4	1.03
32	0.01	0.5	1.03

Table G.9: Validation performance metrics for the four target variables using the Edgeless Graph Architecture (part 1).

Batch Size	LR	Dropout Prob	NRMSE
64	0.00001	0.2	1.13
64	0.00001	0.3	1.35
64	0.00001	0.4	1.49
64	0.00001	0.5	1.42
64	0.0001	0.2	1.02
64	0.0001	0.3	1.02
64	0.0001	0.4	1.02
64	0.0001	0.5	1.47
64	0.001	0.2	0.94
64	0.001	0.3	0.96
64	0.001	0.4	0.94
64	0.001	0.5	1.02
64	0.01	0.2	0.94
64	0.01	0.3	0.97
64	0.01	0.4	1.02
64	0.01	0.5	1.02

Table G.10: Validation performance metrics for the four target variables using the Edgeless Graph Architecture (part 2).

Appendix H

Results for Edgeless GNN Model

Loss Plots

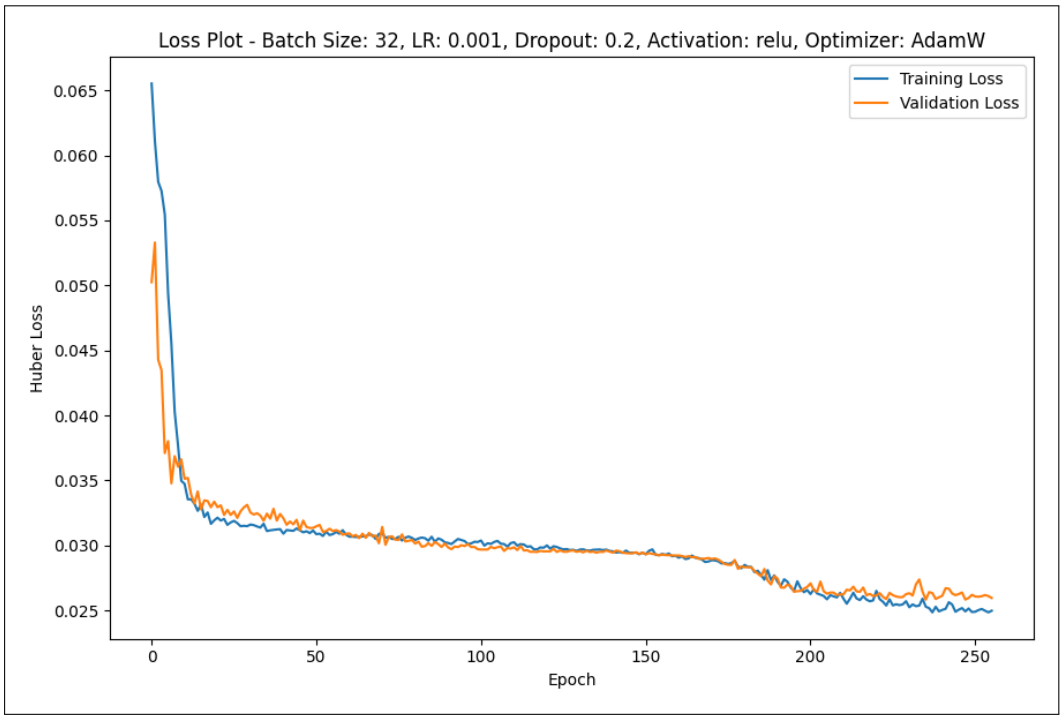

Figure H.1: Loss Plot for Model 3.

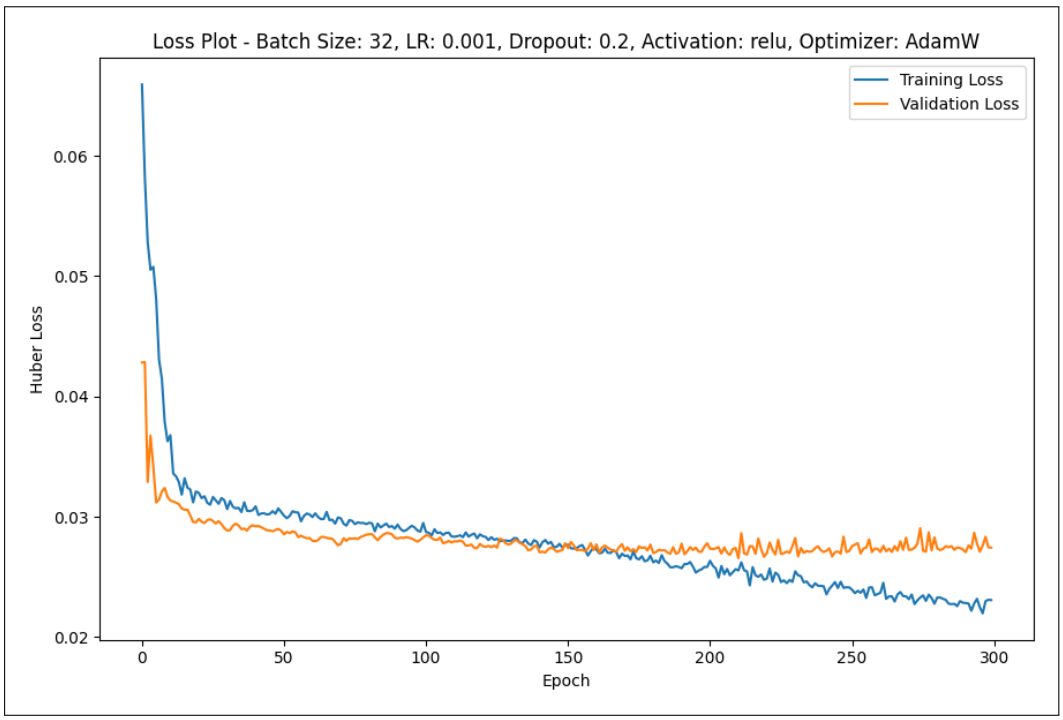

Figure H.2: Loss Plot for Model 7.

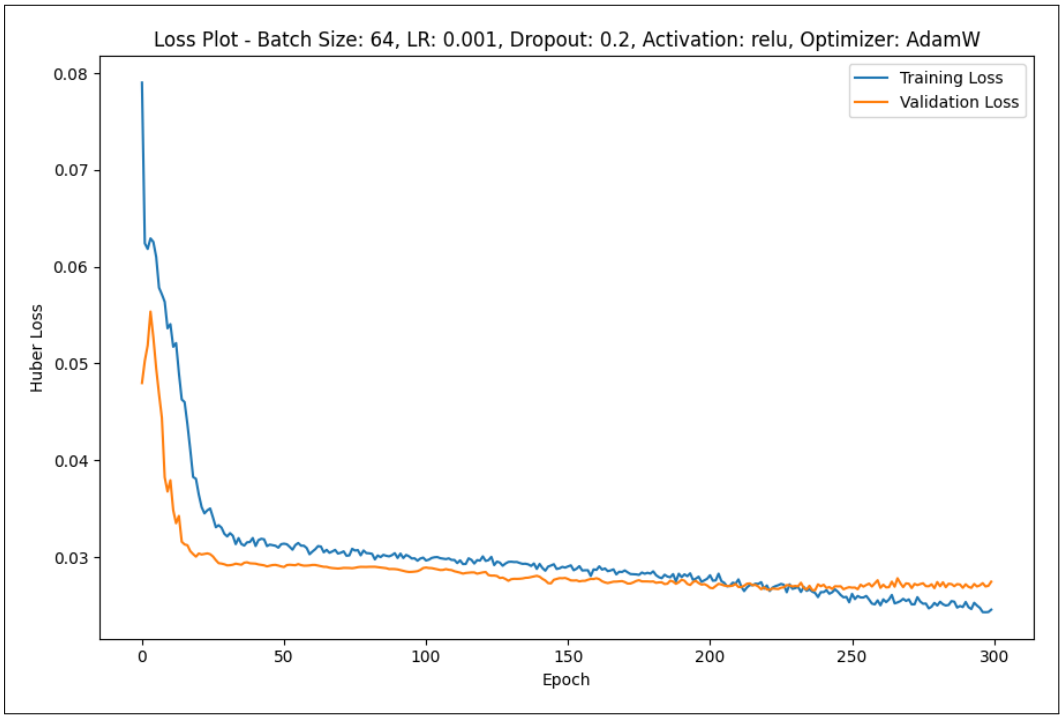

Figure H.3: Loss Plot for Model 8.

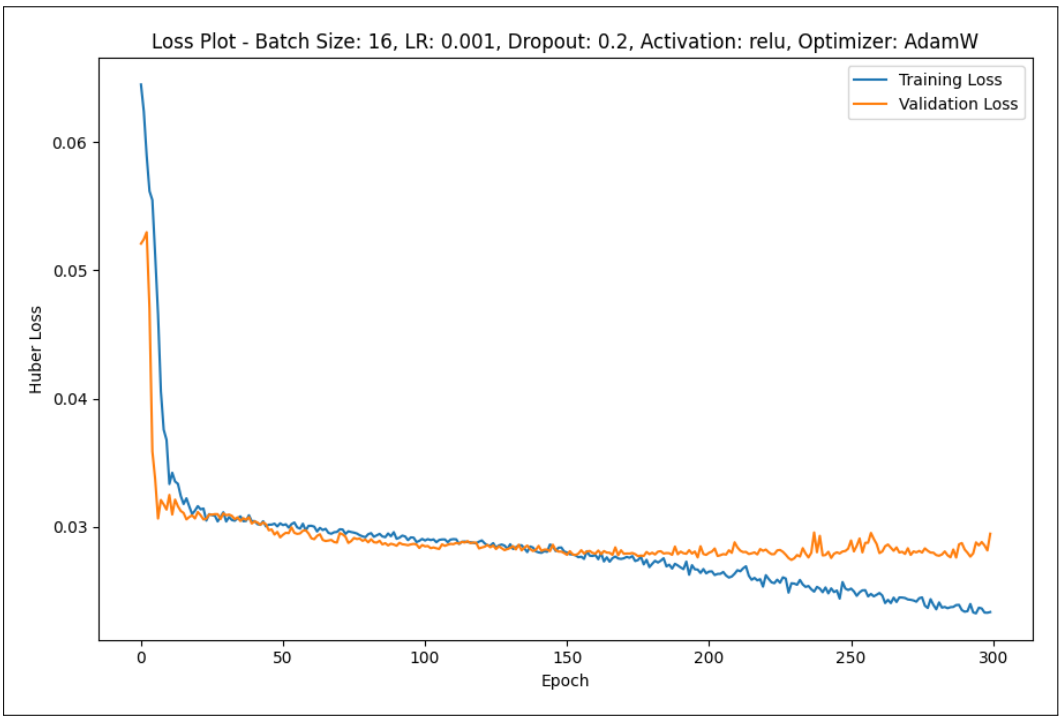

Figure H.4: Loss Plot for Model 9.

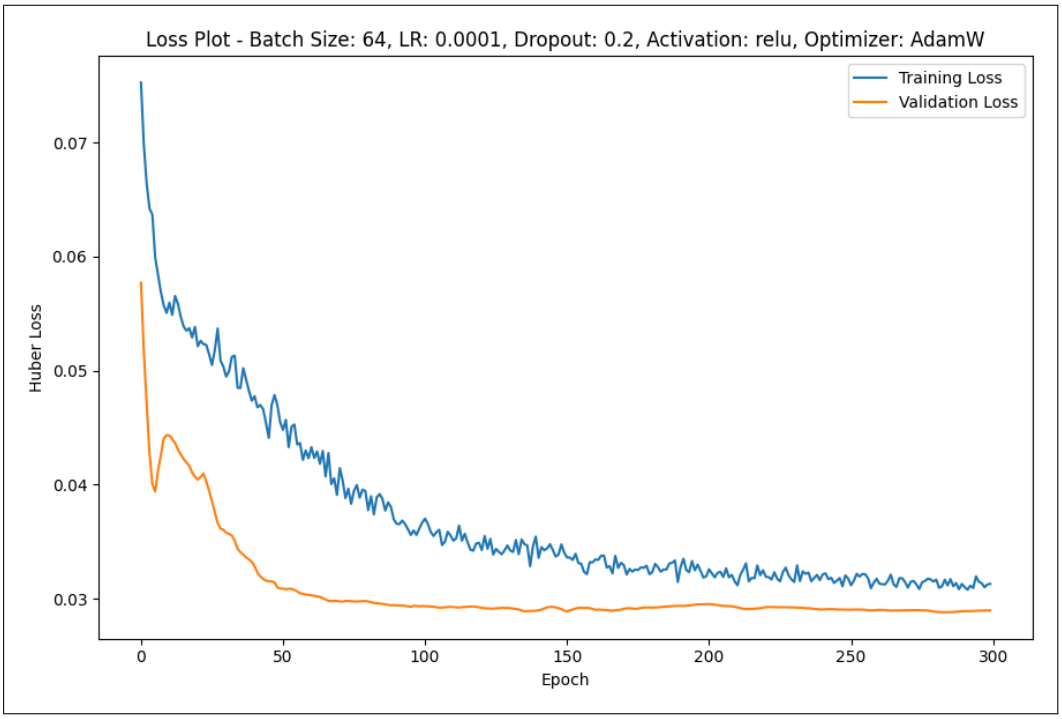

Figure H.5: Loss Plot for Model 10.

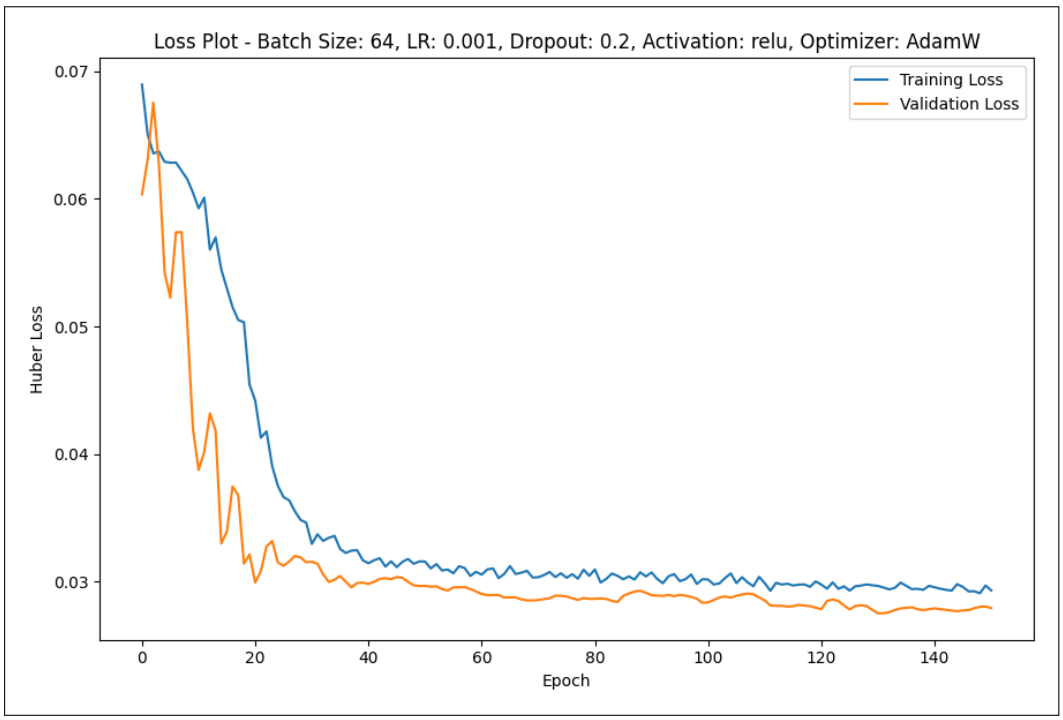

Figure H.6: Loss Plot for Model 11.

Predictive Plots

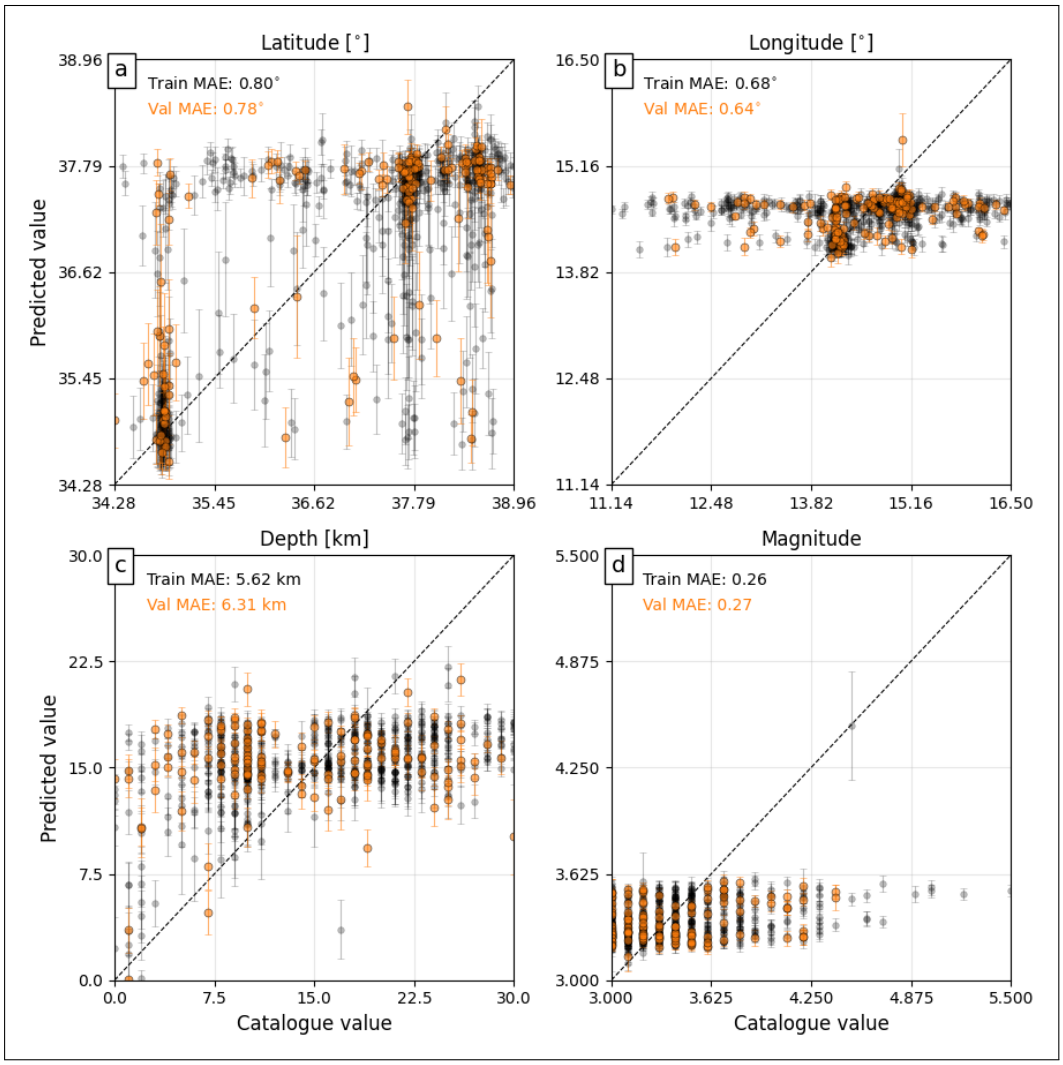

Figure H.7: Actual versus predicted target values for Model 3 with corresponding MAE computed on training and validation sets.

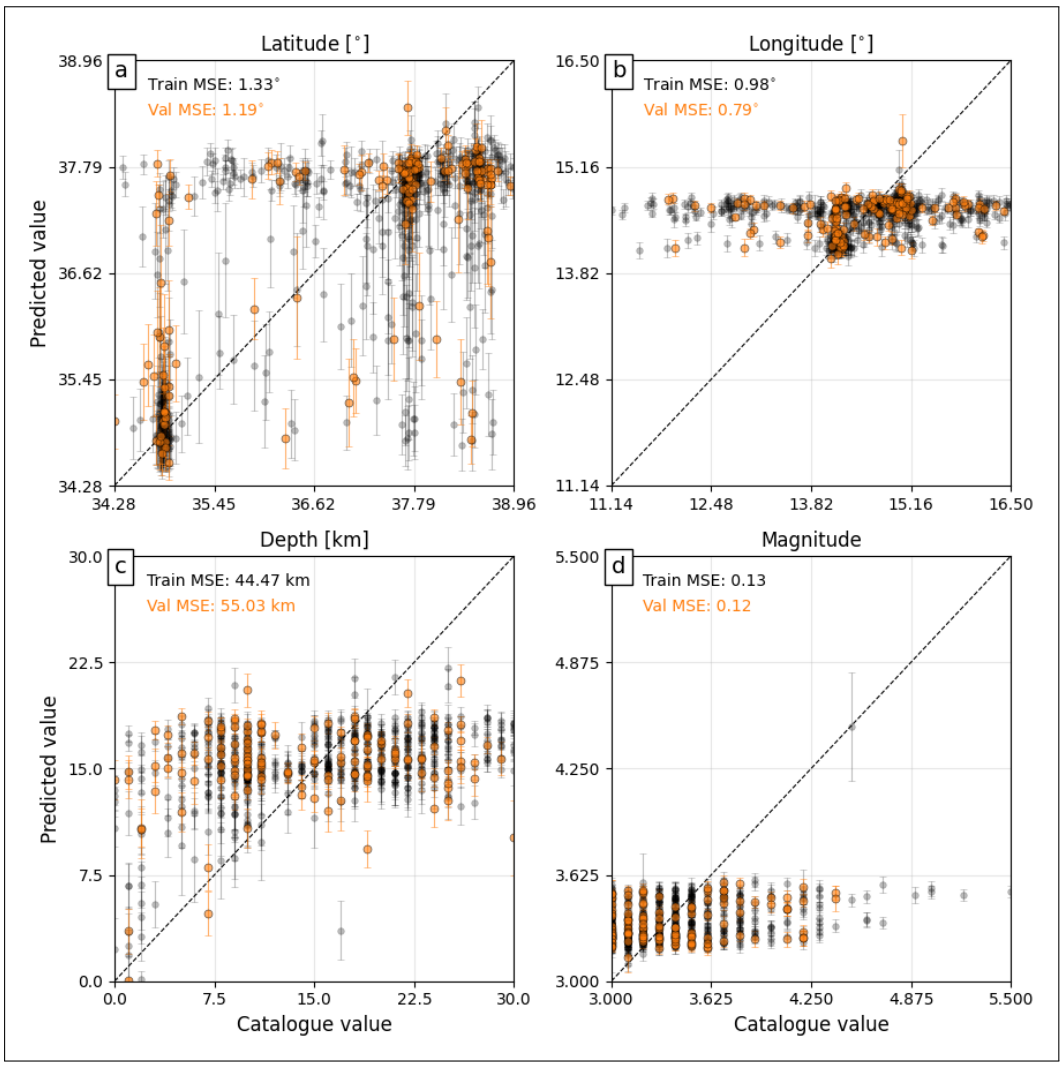

Figure H.8: Actual versus predicted target values for Model 3 with corresponding MSE computed on training and validation sets.

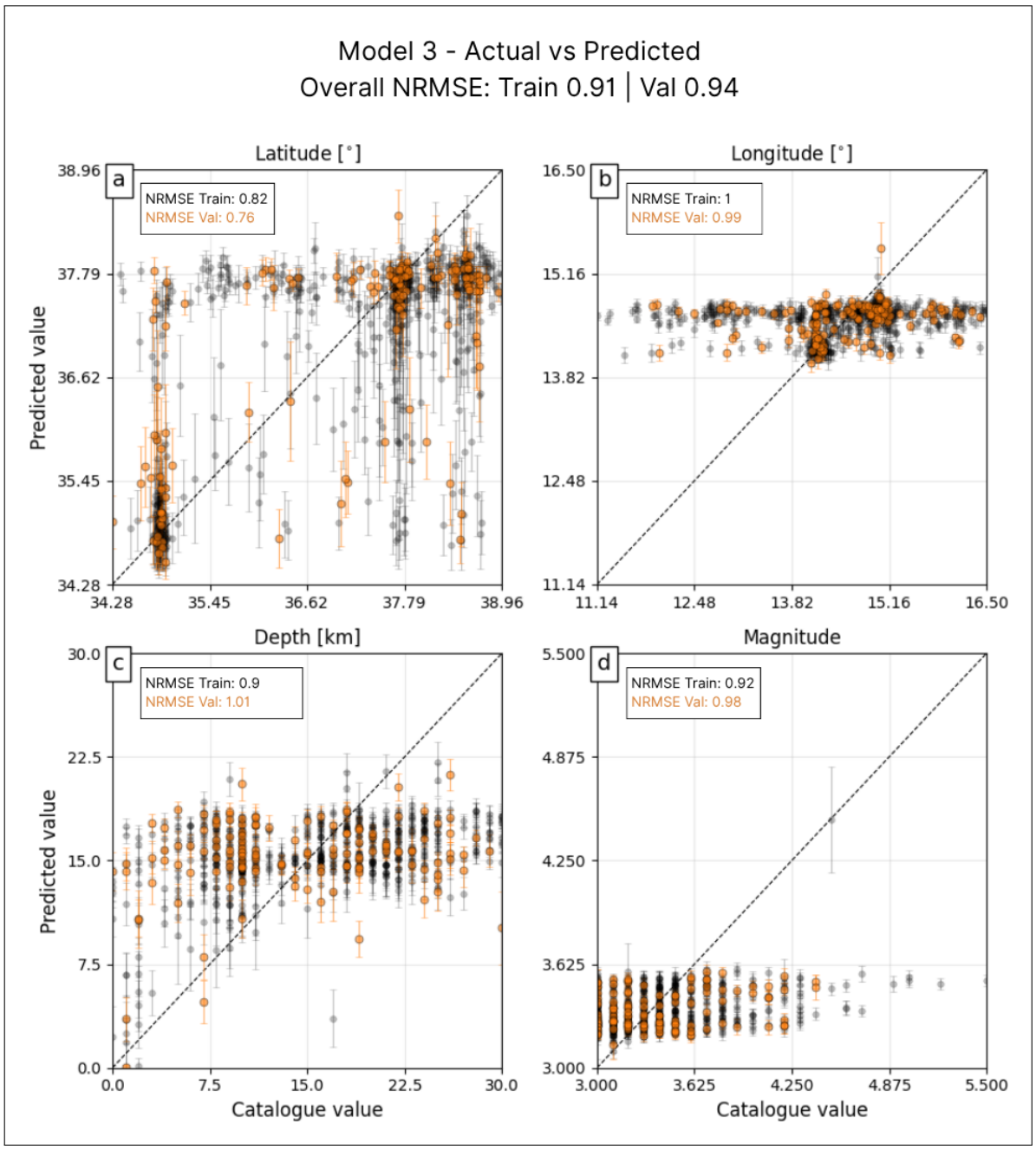

Figure H.9: Actual versus predicted target values for Model 3 with corresponding NRMSE computed on training and validation sets.

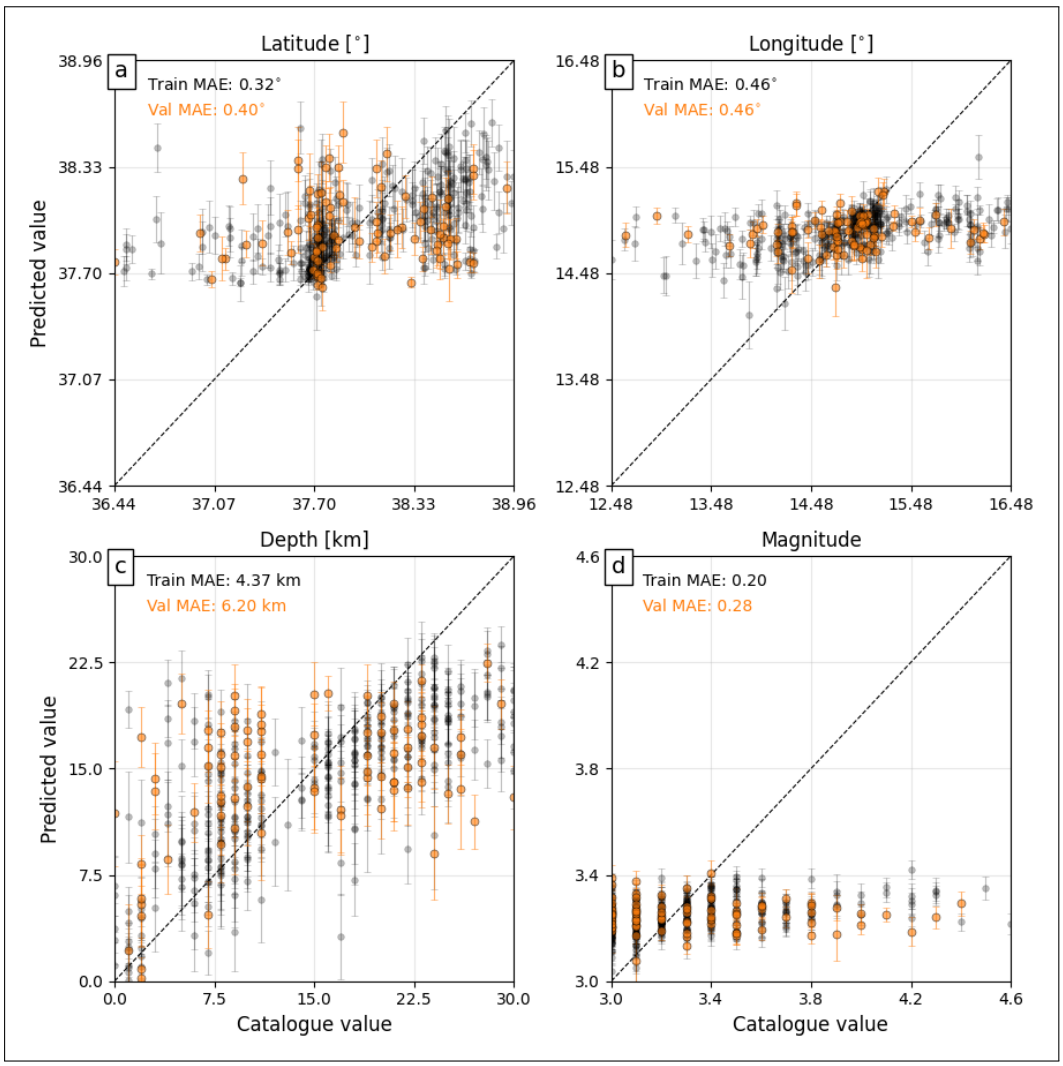

Figure H.10: Actual versus predicted target values for Model 7 with corresponding MAE computed on training and validation sets.

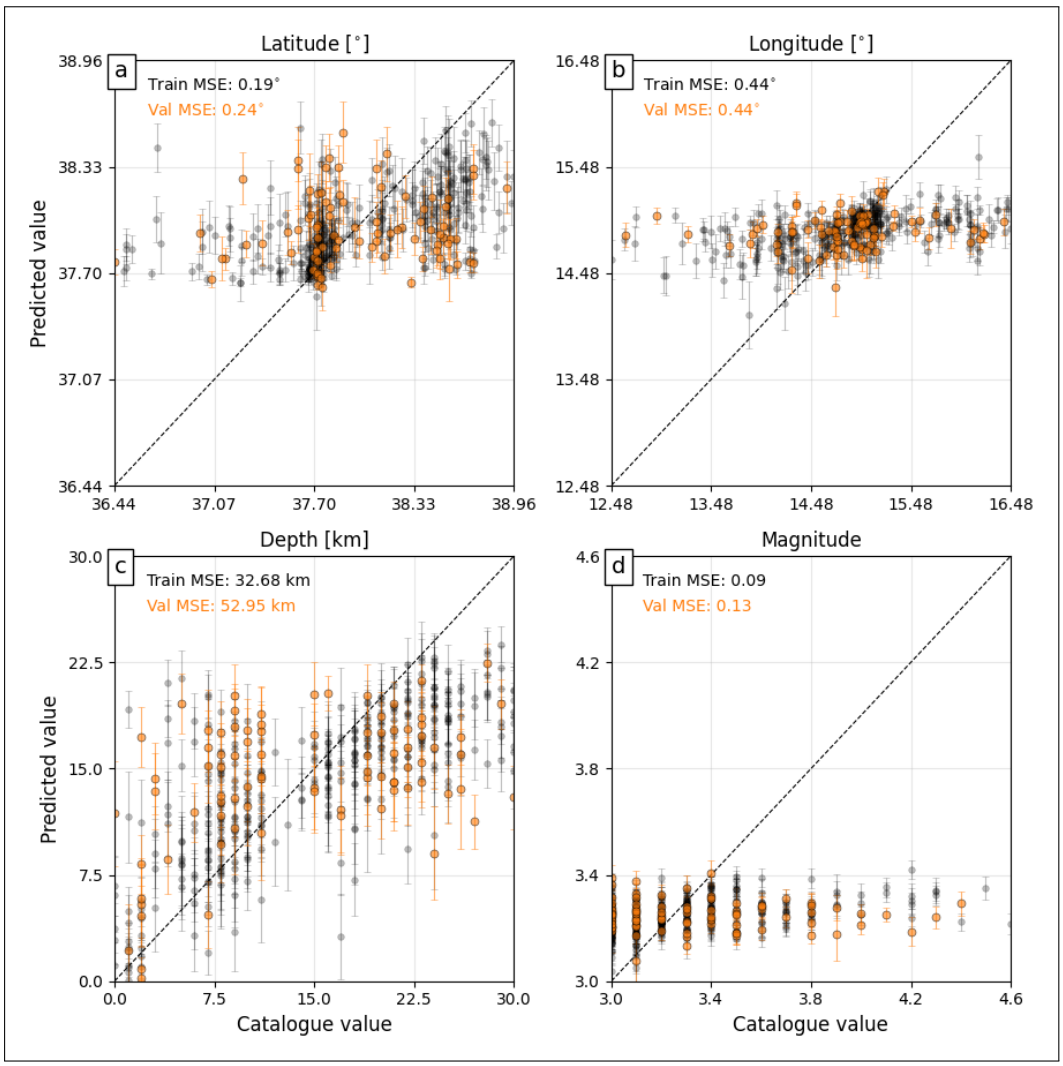

Figure H.11: Actual versus predicted target values for Model 7 with corresponding MSE computed on training and validation sets.

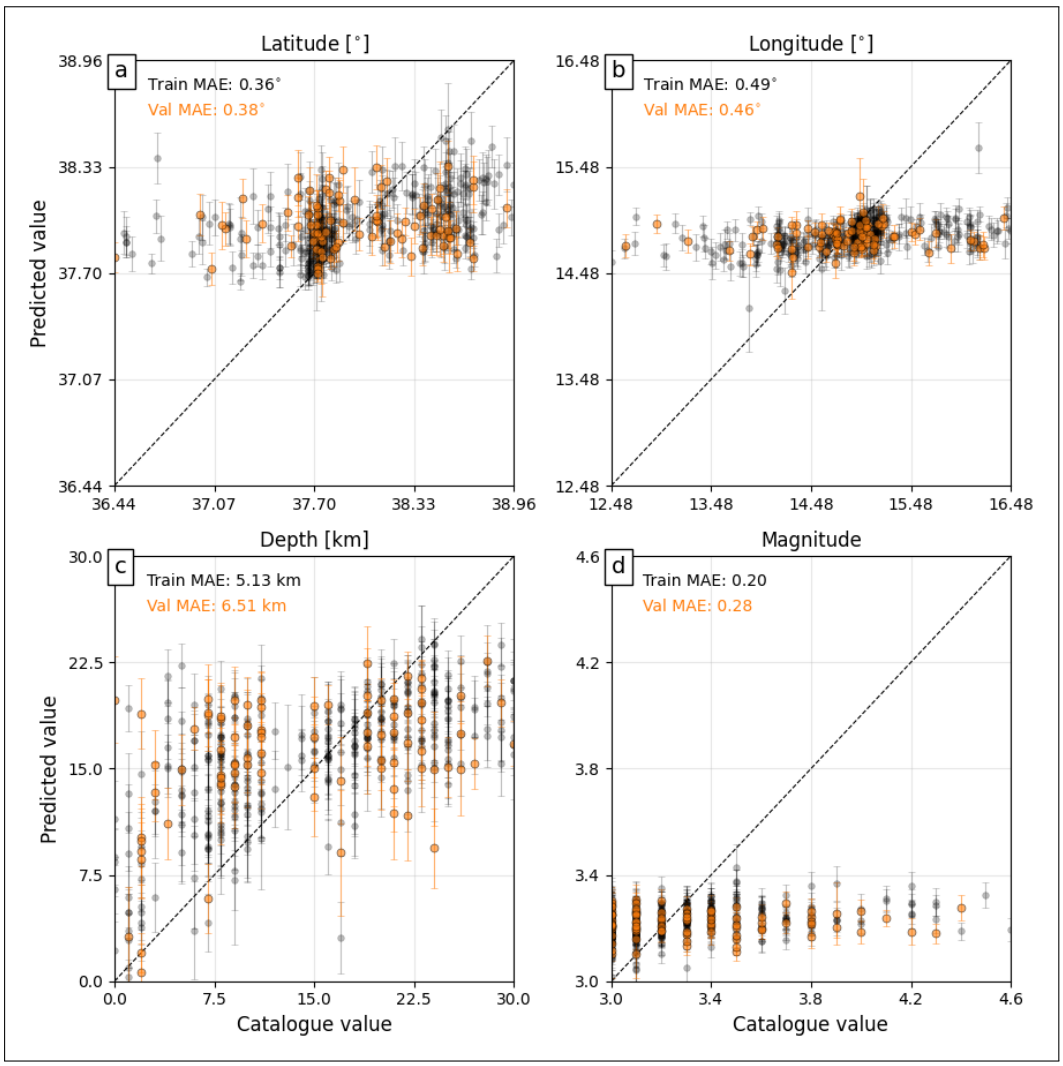

Figure H.12: Actual versus predicted target values for Model 8 with corresponding MAE computed on training and validation sets.

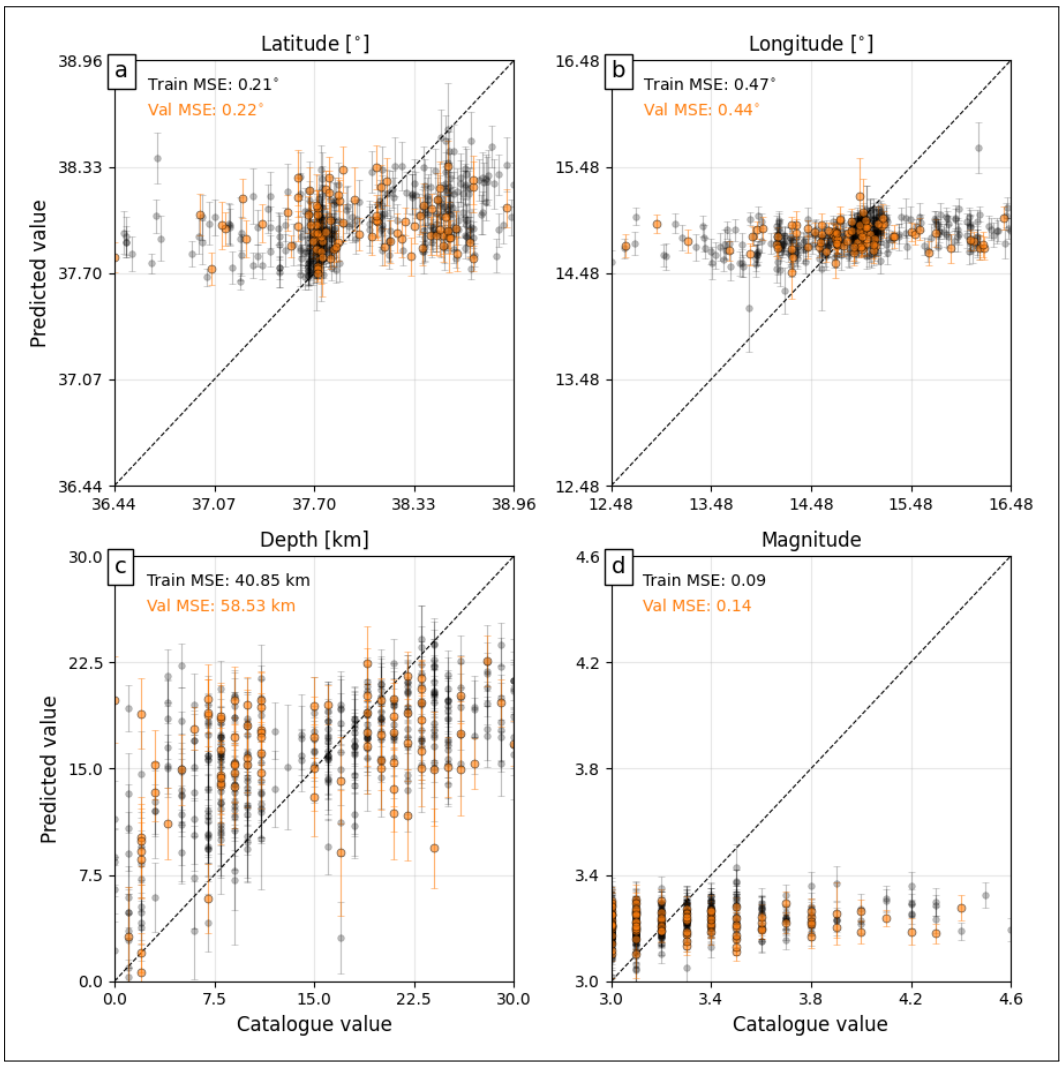

Figure H.13: Actual versus predicted target values for Model 8 with corresponding MSE computed on training and validation sets.

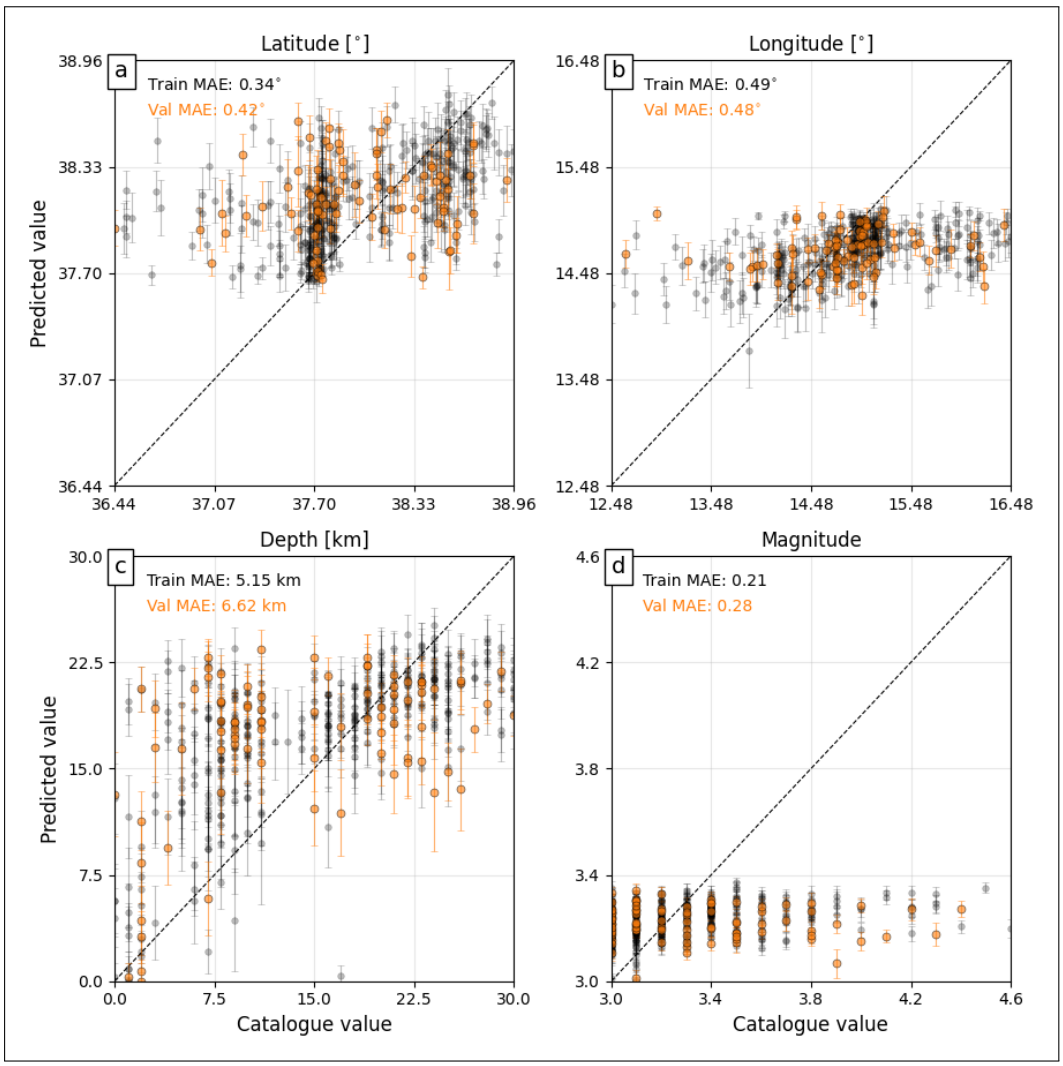

Figure H.14: Actual versus predicted target values for Model 9 with corresponding MAE computed on training and validation sets.

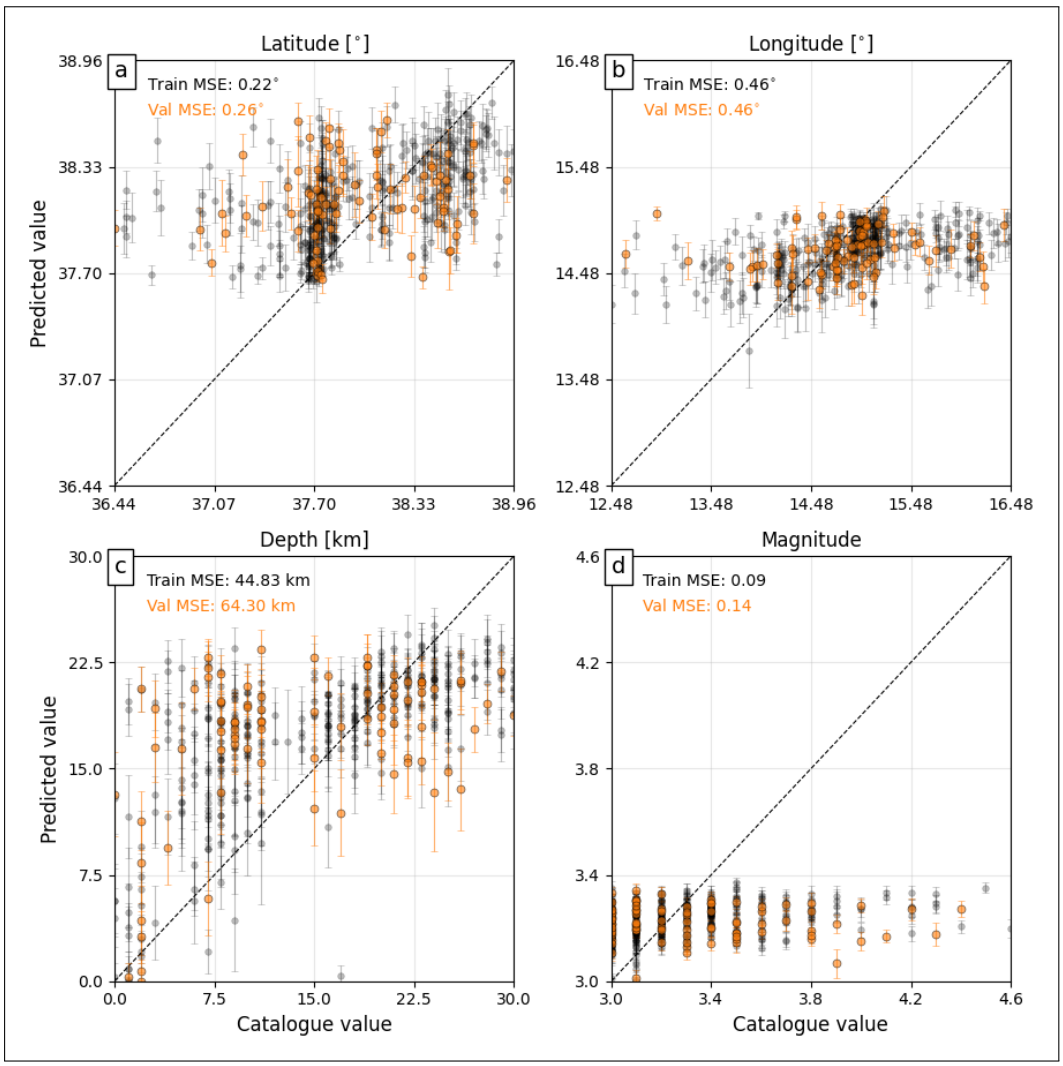

Figure H.15: Actual versus predicted target values for Model 9 with corresponding MSE computed on training and validation sets.

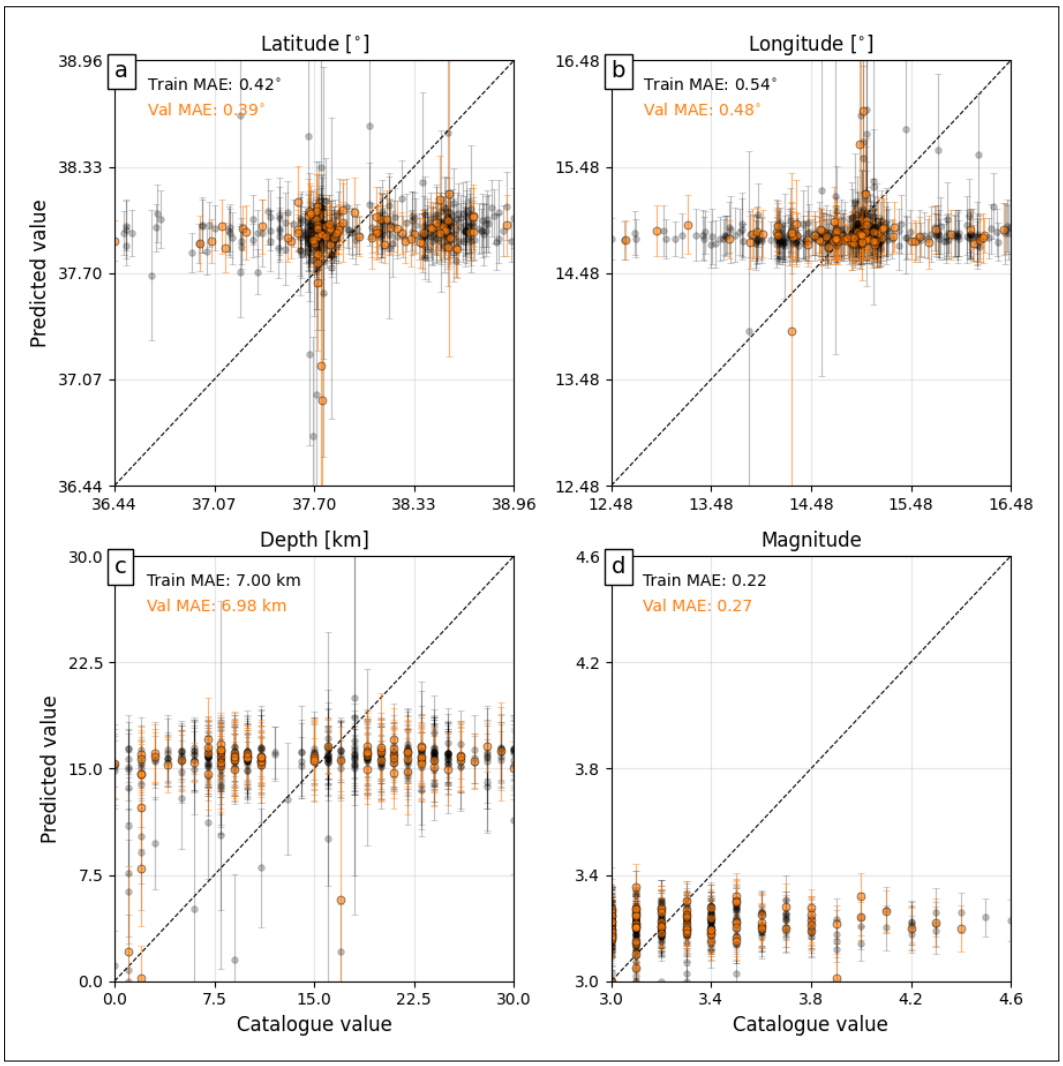

Figure H.16: Actual versus predicted target values for Model 10 with corresponding MAE computed on training and validation sets.

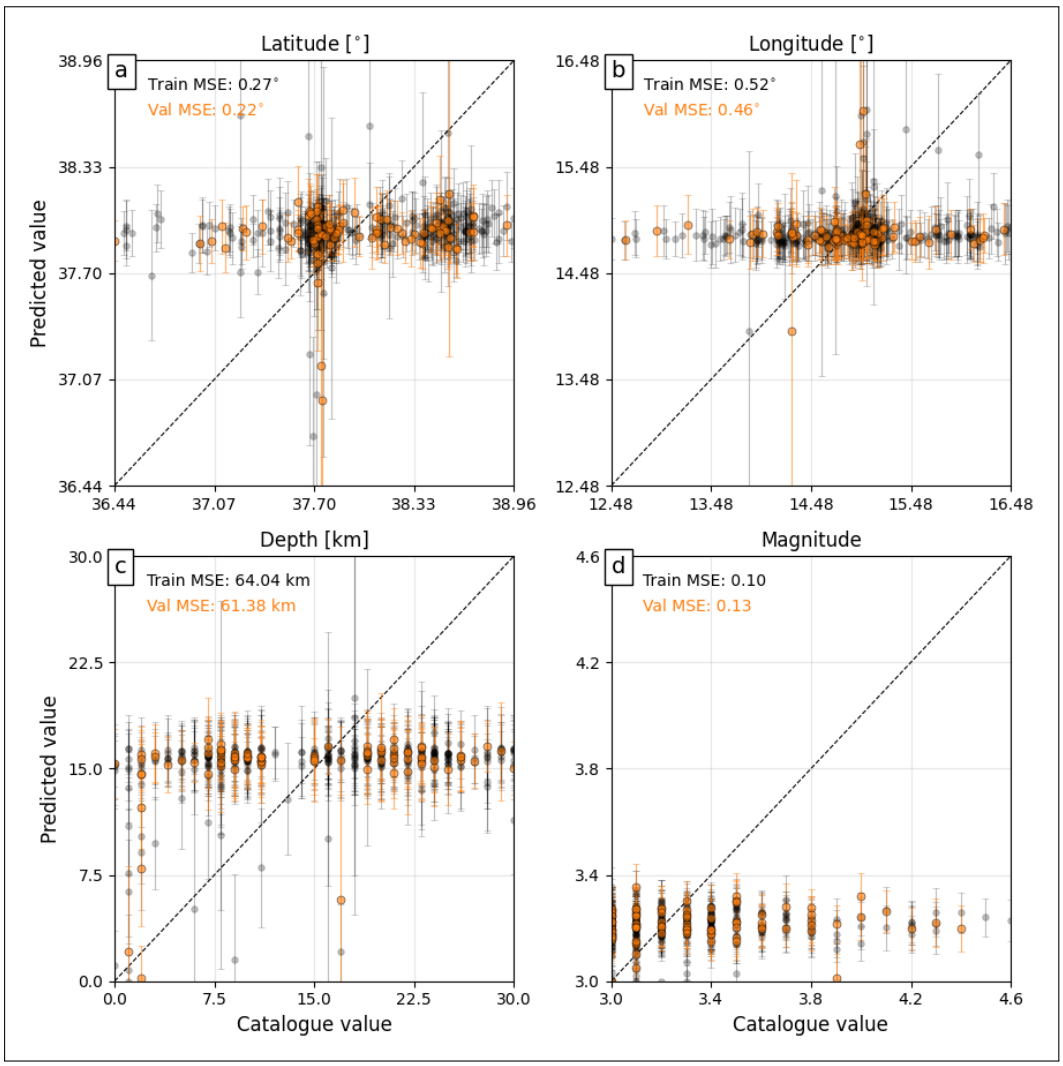

Figure H.17: Actual versus predicted target values for Model 10 with corresponding MSE computed on training and validation sets.

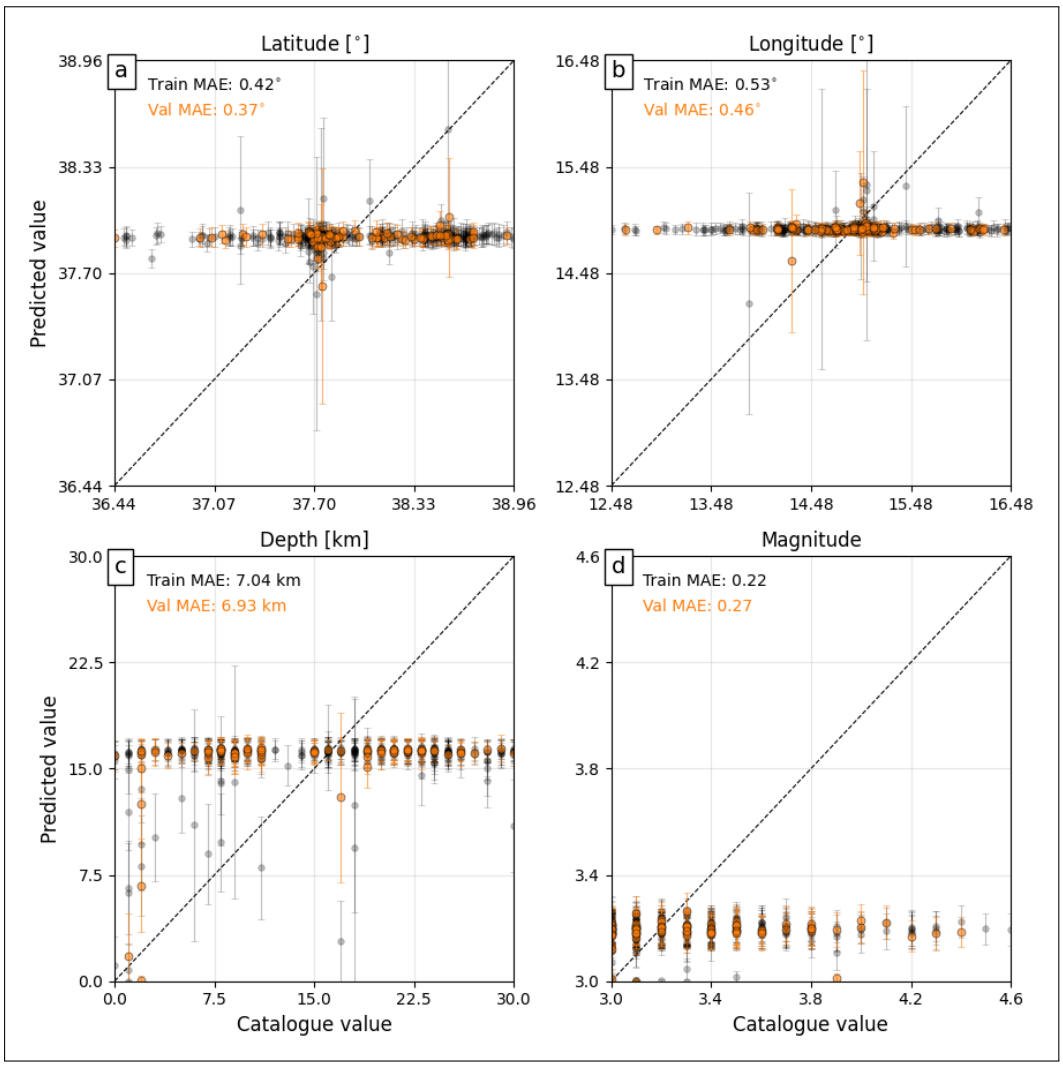

Figure H.18: Actual versus predicted target values for Model 11 with corresponding MAE computed on training and validation sets.

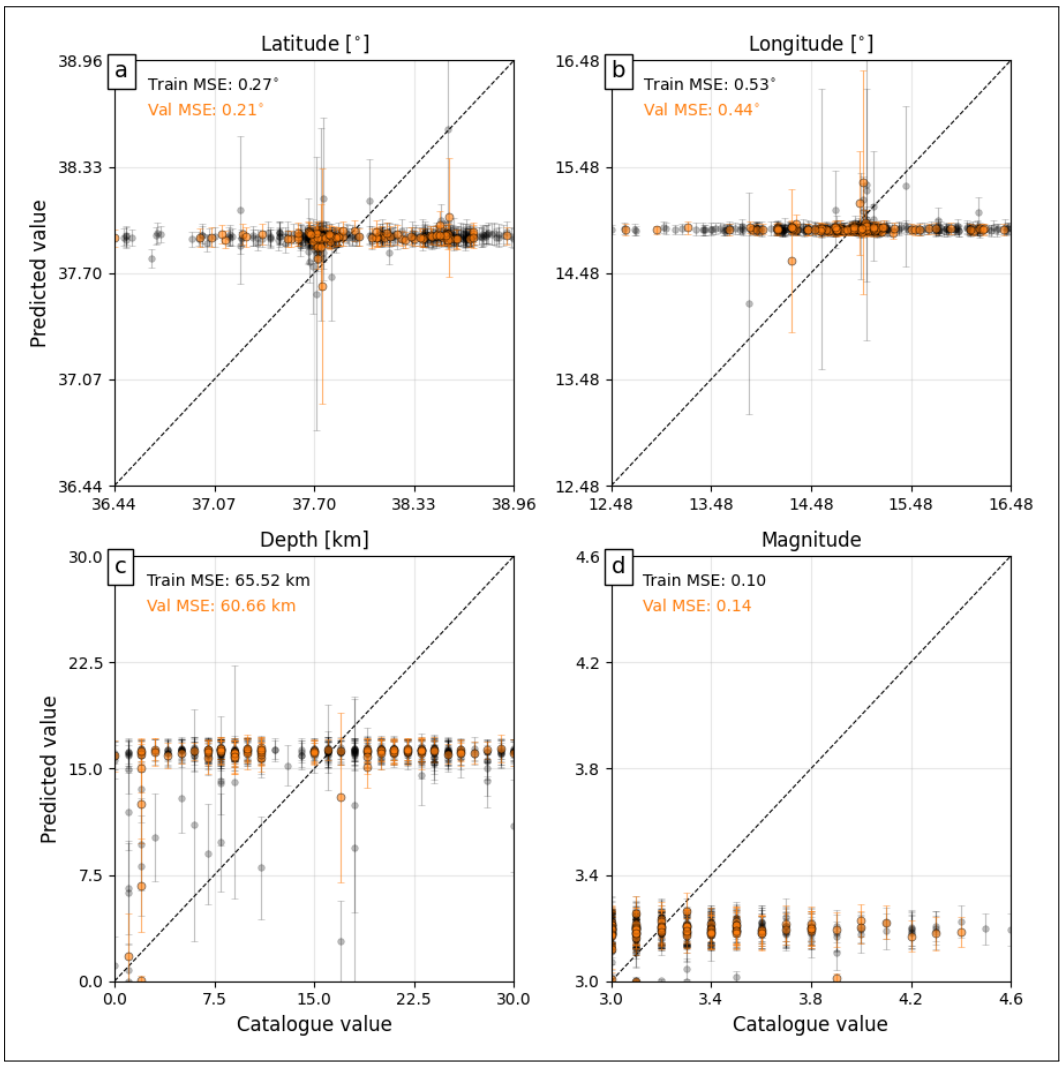

Figure H.19: Actual versus predicted target values for Model 11 with corresponding MSE computed on training and validation sets.

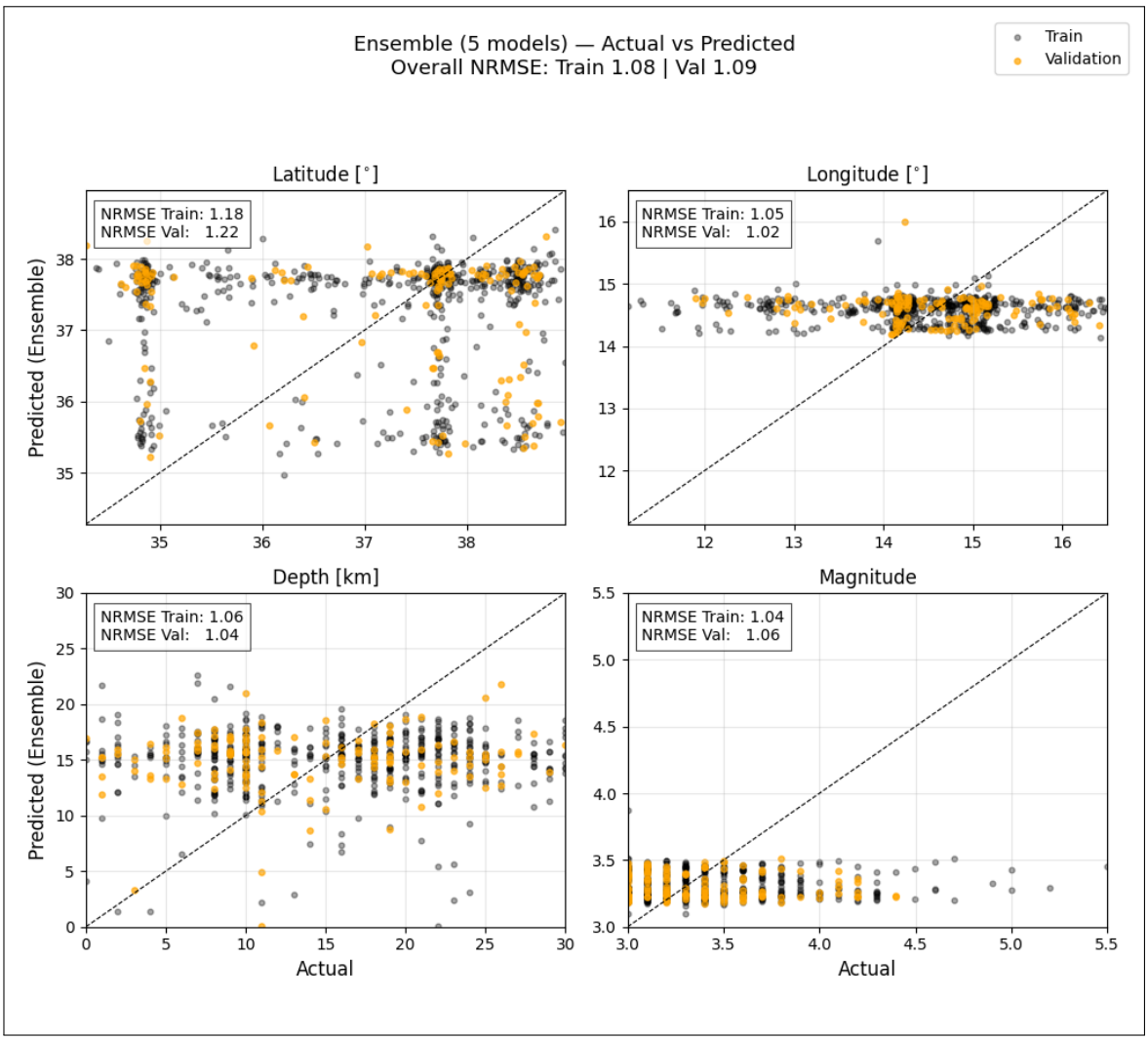

Figure H.20: Actual versus predicted target values for Edgeless Graph Ensemble Model with corresponding NRMSE computed on training and validation sets.

Appendix I

Summary of Performance Metrics for Hyperparameter Tuning using Dynamic Edges GNN Model on Full Dataset

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
3	16	0.00001	0.2	1.76	5.17	1.57
3	16	0.00001	0.3	1.76	5.22	1.58
3	16	0.00001	0.4	1.29	3.24	1.25
3	16	0.00001	0.5	1.32	3.37	1.27
3	16	0.0001	0.2	1.11	2.09	1.00
3	16	0.0001	0.3	1.07	1.94	0.96
3	16	0.0001	0.4	1.82	5.39	1.61
3	16	0.0001	0.5	1.81	5.35	1.60
3	16	0.001	0.2	0.98	1.97	0.97
3	16	0.001	0.3	0.84	1.60	0.88
3	16	0.001	0.4	0.92	1.71	0.91
3	16	0.001	0.5	1.12	2.28	1.05
3	16	0.01	0.2	1.14	2.35	1.06
3	16	0.01	0.3	1.14	2.33	1.06
3	16	0.01	0.4	1.14	2.34	1.06
3	16	0.01	0.5	1.14	2.34	1.06
3	32	0.00001	0.2	1.72	5.03	1.55
3	32	0.00001	0.3	1.56	4.45	1.46
3	32	0.00001	0.4	1.57	4.56	1.48
3	32	0.00001	0.5	1.20	1.95	0.97
3	32	0.0001	0.2	1.01	1.59	0.87
3	32	0.0001	0.3	1.83	5.42	1.61
3	32	0.0001	0.4	1.81	5.36	1.60
3	32	0.0001	0.5	1.81	5.36	1.60
3	32	0.001	0.2	1.01	2.07	1.00
3	32	0.001	0.3	1.17	2.59	1.11
3	32	0.001	0.4	0.88	1.48	0.84
3	32	0.001	0.5	1.13	2.25	1.04
3	32	0.01	0.2	1.03	1.88	0.95
3	32	0.01	0.3	1.14	2.33	1.06
3	32	0.01	0.4	1.14	2.36	1.06
3	32	0.01	0.5	1.14	2.34	1.06

Table I.1: Validation performance metrics for latitude using the Dynamic Edges GNN Architecture (part 1).

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
3	64	0.00001	0.2	1.65	4.77	1.51
3	64	0.00001	0.3	1.51	4.23	1.42
3	64	0.00001	0.4	1.17	2.26	1.04
3	64	0.00001	0.5	1.20	2.31	1.05
3	64	0.0001	0.2	1.09	1.95	0.97
3	64	0.0001	0.3	1.76	5.20	1.58
3	64	0.0001	0.4	1.82	5.40	1.61
3	64	0.0001	0.5	1.78	5.24	1.59
3	64	0.001	0.2	1.04	2.01	0.98
3	64	0.001	0.3	0.96	1.79	0.93
3	64	0.001	0.4	1.09	2.00	0.98
3	64	0.001	0.5	1.14	2.31	1.05
3	64	0.01	0.2	0.79	1.44	0.83
3	64	0.01	0.3	0.94	1.86	0.94
3	64	0.01	0.4	1.15	2.31	1.05
3	64	0.01	0.5	1.14	2.35	1.06
4	16	0.00001	0.2	1.82	5.41	1.61
4	16	0.00001	0.3	1.74	5.11	1.57
4	16	0.00001	0.4	1.35	3.61	1.32
4	16	0.00001	0.5	1.25	3.06	1.21
4	16	0.0001	0.2	1.13	2.21	1.03
4	16	0.0001	0.3	1.14	2.07	1.00
4	16	0.0001	0.4	1.83	5.43	1.61
4	16	0.0001	0.5	1.77	5.22	1.58
4	16	0.001	0.2	0.92	1.73	0.91
4	16	0.001	0.3	1.02	2.06	0.99
4	16	0.001	0.4	0.82	1.41	0.82
4	16	0.001	0.5	0.90	1.58	0.87
4	16	0.01	0.2	1.14	2.32	1.05
4	16	0.01	0.3	1.14	2.33	1.06
4	16	0.01	0.4	1.14	2.32	1.05
4	16	0.01	0.5	1.14	2.35	1.06
4	32	0.00001	0.2	1.77	5.22	1.58
4	32	0.00001	0.3	1.71	4.99	1.55
4	32	0.00001	0.4	1.42	3.87	1.36
4	32	0.00001	0.5	1.26	3.02	1.20
4	32	0.0001	0.2	1.12	2.16	1.02
4	32	0.0001	0.3	1.82	5.40	1.61
4	32	0.0001	0.4	1.82	5.39	1.61
4	32	0.0001	0.5	1.73	5.05	1.56
4	32	0.001	0.2	1.12	2.32	1.05
4	32	0.001	0.3	1.02	2.09	1.00
4	32	0.001	0.4	0.80	1.34	0.80
4	32	0.001	0.5	1.13	2.31	1.05
4	32	0.01	0.2	1.14	2.33	1.06
4	32	0.01	0.3	1.14	2.35	1.06
4	32	0.01	0.4	1.14	2.33	1.06
4	32	0.01	0.5	1.14	2.34	1.06

Table I.2: Validation performance metrics for latitude using the Dynamic Edges GNN Architecture (part 2).

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
4	64	0.00001	0.2	1.38	3.67	1.33
4	64	0.00001	0.3	1.52	4.32	1.44
4	64	0.00001	0.4	1.24	2.26	1.04
4	64	0.00001	0.5	1.37	2.27	1.04
4	64	0.0001	0.2	1.36	3.58	1.31
4	64	0.0001	0.3	1.83	5.44	1.62
4	64	0.0001	0.4	1.78	5.25	1.59
4	64	0.0001	0.5	1.76	5.24	1.59
4	64	0.001	0.2	0.91	1.74	0.91
4	64	0.001	0.3	0.82	1.40	0.82
4	64	0.001	0.4	1.14	2.19	1.02
4	64	0.001	0.5	1.83	5.44	1.62
4	64	0.01	0.2	0.83	1.51	0.85
4	64	0.01	0.3	0.85	1.50	0.85
4	64	0.01	0.4	0.87	1.52	0.85
4	64	0.01	0.5	1.14	2.33	1.06

Table I.3: Validation performance metrics for latitude using the Dynamic Edges GNN Architecture (part 3).

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
3	16	0.00001	0.2	1.92	4.48	2.36
3	16	0.00001	0.3	1.88	4.35	2.32
3	16	0.00001	0.4	1.84	4.21	2.28
3	16	0.00001	0.5	1.39	2.62	1.80
3	16	0.0001	0.2	0.68	0.81	1.00
3	16	0.0001	0.3	0.67	0.81	1.00
3	16	0.0001	0.4	1.99	4.77	2.43
3	16	0.0001	0.5	1.98	4.74	2.42
3	16	0.001	0.2	0.63	0.82	1.01
3	16	0.001	0.3	0.65	0.87	1.04
3	16	0.001	0.4	0.64	0.84	1.02
3	16	0.001	0.5	0.65	0.79	0.99
3	16	0.01	0.2	0.69	0.81	1.00
3	16	0.01	0.3	0.69	0.81	1.00
3	16	0.01	0.4	0.69	0.81	1.00
3	16	0.01	0.5	0.69	0.81	1.00
3	32	0.00001	0.2	1.89	4.38	2.33
3	32	0.00001	0.3	1.93	4.53	2.37
3	32	0.00001	0.4	1.29	2.33	1.70
3	32	0.00001	0.5	1.56	3.19	1.99
3	32	0.0001	0.2	0.67	0.82	1.01
3	32	0.0001	0.3	1.99	4.76	2.43
3	32	0.0001	0.4	1.99	4.75	2.43
3	32	0.0001	0.5	1.96	4.66	2.40
3	32	0.001	0.2	0.66	0.84	1.02
3	32	0.001	0.3	0.66	0.90	1.06
3	32	0.001	0.4	0.62	0.79	0.99
3	32	0.001	0.5	0.68	0.82	1.01
3	32	0.01	0.2	0.68	0.80	1.00
3	32	0.01	0.3	0.69	0.81	1.00
3	32	0.01	0.4	0.69	0.81	1.00
3	32	0.01	0.5	0.69	0.81	1.00

Table I.4: Validation performance metrics for longitude using the Dynamic Edges GNN Architecture (part 1).

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
3	64	0.00001	0.2	1.91	4.46	2.35
3	64	0.00001	0.3	1.90	4.41	2.34
3	64	0.00001	0.4	1.29	2.38	1.72
3	64	0.00001	0.5	0.82	1.16	1.20
3	64	0.0001	0.2	0.69	0.84	1.02
3	64	0.0001	0.3	1.78	3.95	2.21
3	64	0.0001	0.4	1.98	4.75	2.43
3	64	0.0001	0.5	1.94	4.60	2.39
3	64	0.001	0.2	0.73	0.98	1.10
3	64	0.001	0.3	0.64	0.83	1.01
3	64	0.001	0.4	0.67	0.82	1.01
3	64	0.001	0.5	0.69	0.81	1.00
3	64	0.01	0.2	0.64	0.81	1.00
3	64	0.01	0.3	0.68	0.84	1.02
3	64	0.01	0.4	0.69	0.81	1.00
3	64	0.01	0.5	0.69	0.81	1.00
4	16	0.00001	0.2	1.96	4.65	2.40
4	16	0.00001	0.3	1.85	4.22	2.29
4	16	0.00001	0.4	1.15	1.92	1.54
4	16	0.00001	0.5	1.22	2.17	1.64
4	16	0.0001	0.2	0.69	0.81	1.00
4	16	0.0001	0.3	0.70	0.84	1.02
4	16	0.0001	0.4	1.98	4.74	2.42
4	16	0.0001	0.5	1.97	4.67	2.41
4	16	0.001	0.2	0.63	0.80	1.00
4	16	0.001	0.3	0.64	0.81	1.00
4	16	0.001	0.4	0.66	0.84	1.02
4	16	0.001	0.5	0.67	0.85	1.03
4	16	0.01	0.2	0.69	0.81	1.00
4	16	0.01	0.3	0.69	0.81	1.00
4	16	0.01	0.4	0.69	0.81	1.00
4	16	0.01	0.5	0.69	0.81	1.00
4	32	0.00001	0.2	1.97	4.70	2.41
4	32	0.00001	0.3	1.89	4.35	2.32
4	32	0.00001	0.4	1.68	3.64	2.12
4	32	0.00001	0.5	1.70	3.71	2.14
4	32	0.0001	0.2	0.69	0.82	1.01
4	32	0.0001	0.3	1.99	4.77	2.43
4	32	0.0001	0.4	1.98	4.70	2.41
4	32	0.0001	0.5	1.97	4.70	2.41
4	32	0.001	0.2	0.75	1.11	1.17
4	32	0.001	0.3	0.65	0.82	1.01
4	32	0.001	0.4	0.64	0.81	1.00
4	32	0.001	0.5	0.68	0.80	1.00
4	32	0.01	0.2	0.69	0.81	1.00
4	32	0.01	0.3	0.68	0.81	1.00
4	32	0.01	0.4	0.69	0.81	1.00
4	32	0.01	0.5	0.69	0.81	1.00

Table I.5: Validation performance metrics for longitude using the Dynamic Edges GNN Architecture (part 2).

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
4	64	0.00001	0.2	1.79	4.01	2.23
4	64	0.00001	0.3	1.73	3.80	2.17
4	64	0.00001	0.4	1.62	3.40	2.05
4	64	0.00001	0.5	1.23	2.22	1.66
4	64	0.0001	0.2	1.99	4.75	2.43
4	64	0.0001	0.3	1.98	4.72	2.42
4	64	0.0001	0.4	1.95	4.61	2.39
4	64	0.0001	0.5	1.88	4.35	2.32
4	64	0.001	0.2	0.67	0.83	1.01
4	64	0.001	0.3	0.66	0.82	1.01
4	64	0.001	0.4	0.68	0.81	1.00
4	64	0.001	0.5	1.99	4.77	2.43
4	64	0.01	0.2	0.65	0.81	1.00
4	64	0.01	0.3	0.66	0.83	1.01
4	64	0.01	0.4	0.65	0.80	1.00
4	64	0.01	0.5	0.69	0.81	1.00

Table I.6: Validation performance metrics for longitude using the Dynamic Edges GNN Architecture (part 3).

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
3	16	0.00001	0.2	7.10	71.23	1.15
3	16	0.00001	0.3	9.41	125.42	1.53
3	16	0.00001	0.4	7.87	87.15	1.27
3	16	0.00001	0.5	6.98	68.09	1.13
3	16	0.0001	0.2	6.34	53.86	1.00
3	16	0.0001	0.3	6.38	54.40	1.01
3	16	0.0001	0.4	10.63	156.78	1.71
3	16	0.0001	0.5	6.46	57.56	1.04
3	16	0.001	0.2	6.71	70.27	1.14
3	16	0.001	0.3	6.64	66.72	1.12
3	16	0.001	0.4	6.59	62.54	1.08
3	16	0.001	0.5	6.36	53.80	1.00
3	16	0.01	0.2	6.50	56.08	1.02
3	16	0.01	0.3	6.50	56.02	1.02
3	16	0.01	0.4	6.52	56.44	1.03
3	16	0.01	0.5	6.49	55.90	1.02
3	32	0.00001	0.2	6.52	59.36	1.05
3	32	0.00001	0.3	6.87	66.06	1.11
3	32	0.00001	0.4	15.03	279.92	2.28
3	32	0.00001	0.5	7.14	73.80	1.17
3	32	0.0001	0.2	6.33	54.02	1.00
3	32	0.0001	0.3	6.81	74.56	1.18
3	32	0.0001	0.4	6.73	64.98	1.10
3	32	0.0001	0.5	6.63	70.45	1.15
3	32	0.001	0.2	6.42	66.40	1.11
3	32	0.001	0.3	6.50	64.13	1.09
3	32	0.001	0.4	6.44	56.63	1.03
3	32	0.001	0.5	6.33	53.94	1.00
3	32	0.01	0.2	6.42	55.39	1.02
3	32	0.01	0.3	6.49	55.94	1.02
3	32	0.01	0.4	6.49	55.87	1.02
3	32	0.01	0.5	6.48	55.67	1.02

Table I.7: Validation performance metrics for depth using the Dynamic Edges GNN Architecture (part 1).

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
3	64	0.00001	0.2	9.14	120.22	1.50
3	64	0.00001	0.3	8.43	100.59	1.37
3	64	0.00001	0.4	6.43	56.09	1.02
3	64	0.00001	0.5	6.29	68.25	1.13
3	64	0.0001	0.2	6.32	53.44	1.00
3	64	0.0001	0.3	7.06	70.05	1.14
3	64	0.0001	0.4	13.01	222.33	2.04
3	64	0.0001	0.5	6.65	58.94	1.05
3	64	0.001	0.2	6.91	69.40	1.14
3	64	0.001	0.3	6.38	58.86	1.05
3	64	0.001	0.4	6.34	53.78	1.00
3	64	0.001	0.5	6.51	56.30	1.02
3	64	0.01	0.2	6.40	55.91	1.02
3	64	0.01	0.3	6.41	55.22	1.01
3	64	0.01	0.4	6.40	55.28	1.02
3	64	0.01	0.5	6.52	56.59	1.03
4	16	0.00001	0.2	7.22	72.08	1.16
4	16	0.00001	0.3	11.84	189.63	1.88
4	16	0.00001	0.4	7.70	83.18	1.25
4	16	0.00001	0.5	10.77	165.67	1.76
4	16	0.0001	0.2	6.38	54.05	1.00
4	16	0.0001	0.3	6.31	53.94	1.00
4	16	0.0001	0.4	6.26	54.92	1.01
4	16	0.0001	0.5	9.64	131.25	1.56
4	16	0.001	0.2	7.21	77.75	1.20
4	16	0.001	0.3	7.22	82.25	1.24
4	16	0.001	0.4	6.59	60.78	1.06
4	16	0.001	0.5	6.33	55.81	1.02
4	16	0.01	0.2	6.53	56.69	1.03
4	16	0.01	0.3	6.51	56.24	1.02
4	16	0.01	0.4	6.48	55.55	1.02
4	16	0.01	0.5	6.53	56.72	1.03
4	32	0.00001	0.2	6.48	56.35	1.02
4	32	0.00001	0.3	6.40	57.53	1.04
4	32	0.00001	0.4	6.89	76.14	1.19
4	32	0.00001	0.5	6.44	56.62	1.03
4	32	0.0001	0.2	6.32	53.07	0.99
4	32	0.0001	0.3	12.19	197.93	1.92
4	32	0.0001	0.4	7.27	75.91	1.19
4	32	0.0001	0.5	6.15	59.99	1.06
4	32	0.001	0.2	6.82	76.17	1.19
4	32	0.001	0.3	6.55	60.77	1.06
4	32	0.001	0.4	6.22	55.37	1.02
4	32	0.001	0.5	6.23	52.46	0.99
4	32	0.01	0.2	6.49	55.86	1.02
4	32	0.01	0.3	6.21	51.93	0.98
4	32	0.01	0.4	6.52	56.61	1.03
4	32	0.01	0.5	6.54	56.95	1.03

Table I.8: Validation performance metrics for depth using the Dynamic Edges GNN Architecture (part 2).

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
4	64	0.00001	0.2	12.07	195.41	1.91
4	64	0.00001	0.3	6.47	57.13	1.03
4	64	0.00001	0.4	6.59	59.45	1.05
4	64	0.00001	0.5	6.84	63.31	1.09
4	64	0.0001	0.2	6.88	69.70	1.14
4	64	0.0001	0.3	13.36	231.39	2.08
4	64	0.0001	0.4	6.89	76.96	1.20
4	64	0.0001	0.5	7.49	77.52	1.20
4	64	0.001	0.2	6.51	62.83	1.08
4	64	0.001	0.3	6.27	55.19	1.01
4	64	0.001	0.4	6.36	54.05	1.00
4	64	0.001	0.5	15.85	305.19	2.38
4	64	0.01	0.2	6.43	56.07	1.02
4	64	0.01	0.3	6.48	58.32	1.04
4	64	0.01	0.4	6.38	54.44	1.01
4	64	0.01	0.5	6.53	56.79	1.03

Table I.9: Validation performance metrics for depth using the Dynamic Edges GNN Architecture (part 3).

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
3	16	0.00001	0.2	0.37	0.26	1.44
3	16	0.00001	0.3	0.36	0.25	1.41
3	16	0.00001	0.4	0.35	0.24	1.38
3	16	0.00001	0.5	0.33	0.21	1.29
3	16	0.0001	0.2	0.28	0.13	1.02
3	16	0.0001	0.3	0.28	0.13	1.02
3	16	0.0001	0.4	0.37	0.26	1.44
3	16	0.0001	0.5	0.36	0.26	1.44
3	16	0.001	0.2	0.28	0.12	0.98
3	16	0.001	0.3	0.28	0.12	0.98
3	16	0.001	0.4	0.29	0.13	1.02
3	16	0.001	0.5	0.28	0.13	1.02
3	16	0.01	0.2	0.28	0.13	1.02
3	16	0.01	0.3	0.28	0.13	1.02
3	16	0.01	0.4	0.28	0.13	1.02
3	16	0.01	0.5	0.28	0.13	1.02
3	32	0.00001	0.2	0.36	0.26	1.44
3	32	0.00001	0.3	0.36	0.26	1.44
3	32	0.00001	0.4	0.36	0.25	1.41
3	32	0.00001	0.5	0.32	0.20	1.26
3	32	0.0001	0.2	0.28	0.13	1.02
3	32	0.0001	0.3	0.37	0.26	1.44
3	32	0.0001	0.4	0.37	0.26	1.44
3	32	0.0001	0.5	0.37	0.26	1.44
3	32	0.001	0.2	0.28	0.13	1.02
3	32	0.001	0.3	0.28	0.12	0.98
3	32	0.001	0.4	0.28	0.12	0.98
3	32	0.001	0.5	0.28	0.13	1.02
3	32	0.01	0.2	0.28	0.13	1.02
3	32	0.01	0.3	0.28	0.13	1.02
3	32	0.01	0.4	0.28	0.13	1.02
3	32	0.01	0.5	0.28	0.13	1.02

Table I.10: Validation performance metrics for magnitude using the Dynamic Edges GNN Architecture (part 1).

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
3	64	0.00001	0.2	0.37	0.26	1.44
3	64	0.00001	0.3	0.36	0.25	1.41
3	64	0.00001	0.4	0.33	0.22	1.32
3	64	0.00001	0.5	0.29	0.17	1.16
3	64	0.0001	0.2	0.28	0.13	1.02
3	64	0.0001	0.3	0.36	0.26	1.44
3	64	0.0001	0.4	0.37	0.26	1.44
3	64	0.0001	0.5	0.36	0.25	1.41
3	64	0.001	0.2	0.28	0.12	0.98
3	64	0.001	0.3	0.28	0.13	1.02
3	64	0.001	0.4	0.28	0.13	1.02
3	64	0.001	0.5	0.28	0.13	1.02
3	64	0.01	0.2	0.28	0.14	1.06
3	64	0.01	0.3	0.29	0.13	1.02
3	64	0.01	0.4	0.28	0.13	1.02
3	64	0.01	0.5	0.28	0.13	1.02
4	16	0.00001	0.2	0.37	0.26	1.44
4	16	0.00001	0.3	0.36	0.26	1.44
4	16	0.00001	0.4	0.35	0.24	1.38
4	16	0.00001	0.5	0.32	0.20	1.26
4	16	0.0001	0.2	0.28	0.13	1.02
4	16	0.0001	0.3	0.27	0.13	1.02
4	16	0.0001	0.4	0.37	0.26	1.44
4	16	0.0001	0.5	0.37	0.26	1.44
4	16	0.001	0.2	0.28	0.12	0.98
4	16	0.001	0.3	0.29	0.13	1.02
4	16	0.001	0.4	0.27	0.13	1.02
4	16	0.001	0.5	0.28	0.13	1.02
4	16	0.01	0.2	0.28	0.13	1.02
4	16	0.01	0.3	0.28	0.13	1.02
4	16	0.01	0.4	0.28	0.13	1.02
4	16	0.01	0.5	0.28	0.13	1.02
4	32	0.00001	0.2	0.37	0.26	1.44
4	32	0.00001	0.3	0.36	0.26	1.44
4	32	0.00001	0.4	0.34	0.22	1.32
4	32	0.00001	0.5	0.32	0.20	1.26
4	32	0.0001	0.2	0.27	0.13	1.02
4	32	0.0001	0.3	0.37	0.26	1.44
4	32	0.0001	0.4	0.37	0.26	1.44
4	32	0.0001	0.5	0.37	0.26	1.44
4	32	0.001	0.2	0.30	0.14	1.06
4	32	0.001	0.3	0.28	0.13	1.02
4	32	0.001	0.4	0.28	0.13	1.02
4	32	0.001	0.5	0.28	0.13	1.02
4	32	0.01	0.2	0.28	0.13	1.02
4	32	0.01	0.3	0.28	0.13	1.02
4	32	0.01	0.4	0.28	0.13	1.02
4	32	0.01	0.5	0.28	0.13	1.02

Table I.11: Validation performance metrics for magnitude using the Dynamic Edges GNN Architecture (part 2).

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
4	64	0.00001	0.2	0.36	0.25	1.41
4	64	0.00001	0.3	0.36	0.25	1.41
4	64	0.00001	0.4	0.34	0.22	1.32
4	64	0.00001	0.5	0.29	0.16	1.13
4	64	0.0001	0.2	0.36	0.26	1.44
4	64	0.0001	0.3	0.37	0.26	1.44
4	64	0.0001	0.4	0.36	0.26	1.44
4	64	0.0001	0.5	0.36	0.26	1.44
4	64	0.001	0.2	0.28	0.13	1.02
4	64	0.001	0.3	0.28	0.13	1.02
4	64	0.001	0.4	0.27	0.13	1.02
4	64	0.001	0.5	0.37	0.26	1.44
4	64	0.01	0.2	0.27	0.12	0.98
4	64	0.01	0.3	0.28	0.13	1.02
4	64	0.01	0.4	0.27	0.13	1.02
4	64	0.01	0.5	0.28	0.13	1.02

Table I.12: Validation performance metrics for magnitude using the Dynamic Edges GNN Architecture (part 3).

k	Batch Size	LR	Dropout Prob	NRMSE
3	16	0.00001	0.2	1.63
3	16	0.00001	0.3	1.71
3	16	0.00001	0.4	1.55
3	16	0.00001	0.5	1.37
3	16	0.0001	0.2	1.01
3	16	0.0001	0.3	1.00
3	16	0.0001	0.4	1.80
3	16	0.0001	0.5	1.63
3	16	0.001	0.2	1.03
3	16	0.001	0.3	1.00
3	16	0.001	0.4	1.01
3	16	0.001	0.5	1.01
3	16	0.01	0.2	1.03
3	16	0.01	0.3	1.02
3	16	0.01	0.4	1.03
3	16	0.01	0.5	1.03
3	32	0.00001	0.2	1.59
3	32	0.00001	0.3	1.60
3	32	0.00001	0.4	1.72
3	32	0.00001	0.5	1.35
3	32	0.0001	0.2	0.98
3	32	0.0001	0.3	1.67
3	32	0.0001	0.4	1.64
3	32	0.0001	0.5	1.65
3	32	0.001	0.2	1.04
3	32	0.001	0.3	1.06
3	32	0.001	0.4	0.96
3	32	0.001	0.5	1.02
3	32	0.01	0.2	1.00
3	32	0.01	0.3	1.02
3	32	0.01	0.4	1.03
3	32	0.01	0.5	1.02

Table I.13: Validation performance metrics for the four target variables using the Dynamic Edges GNN Architecture (part 1).

k	Batch Size	LR	Dropout Prob	NRMSE
3	64	0.00001	0.2	1.70
3	64	0.00001	0.3	1.64
3	64	0.00001	0.4	1.28
3	64	0.00001	0.5	1.14
3	64	0.0001	0.2	1.00
3	64	0.0001	0.3	1.59
3	64	0.0001	0.4	1.88
3	64	0.0001	0.5	1.61
3	64	0.001	0.2	1.05
3	64	0.001	0.3	1.00
3	64	0.001	0.4	1.00
3	64	0.001	0.5	1.02
3	64	0.01	0.2	0.98
3	64	0.01	0.3	1.00
3	64	0.01	0.4	1.02
3	64	0.01	0.5	1.03
4	16	0.00001	0.2	1.65
4	16	0.00001	0.3	1.79
4	16	0.00001	0.4	1.37
4	16	0.00001	0.5	1.47
4	16	0.0001	0.2	1.01
4	16	0.0001	0.3	1.01
4	16	0.0001	0.4	1.62
4	16	0.0001	0.5	1.75
4	16	0.001	0.2	1.02
4	16	0.001	0.3	1.06
4	16	0.001	0.4	0.98
4	16	0.001	0.5	0.98
4	16	0.01	0.2	1.03
4	16	0.01	0.3	1.03
4	16	0.01	0.4	1.02
4	16	0.01	0.5	1.03
4	32	0.00001	0.2	1.62
4	32	0.00001	0.3	1.59
4	32	0.00001	0.4	1.50
4	32	0.00001	0.5	1.41
4	32	0.0001	0.2	1.01
4	32	0.0001	0.3	1.85
4	32	0.0001	0.4	1.66
4	32	0.0001	0.5	1.62
4	32	0.001	0.2	1.12
4	32	0.001	0.3	1.02
4	32	0.001	0.4	0.96
4	32	0.001	0.5	1.01
4	32	0.01	0.2	1.02
4	32	0.01	0.3	1.02
4	32	0.01	0.4	1.03
4	32	0.01	0.5	1.03

Table I.14: Validation performance metrics for the four target variables using the Dynamic Edges GNN Architecture (part 2).

k	Batch Size	LR	Dropout Prob	NRMSE
4	64	0.00001	0.2	1.72
4	64	0.00001	0.3	1.51
4	64	0.00001	0.4	1.37
4	64	0.00001	0.5	1.23
4	64	0.0001	0.2	1.58
4	64	0.0001	0.3	1.89
4	64	0.0001	0.4	1.65
4	64	0.0001	0.5	1.64
4	64	0.001	0.2	1.01
4	64	0.001	0.3	0.97
4	64	0.001	0.4	1.01
4	64	0.001	0.5	1.97
4	64	0.01	0.2	0.96
4	64	0.01	0.3	0.98
4	64	0.01	0.4	0.97
4	64	0.01	0.5	1.03

Table I.15: Validation performance metrics for the four target variables using the Dynamic Edges GNN Architecture (part 3).

Appendix J

Results for Dynamic Edges GNN

Loss Plots

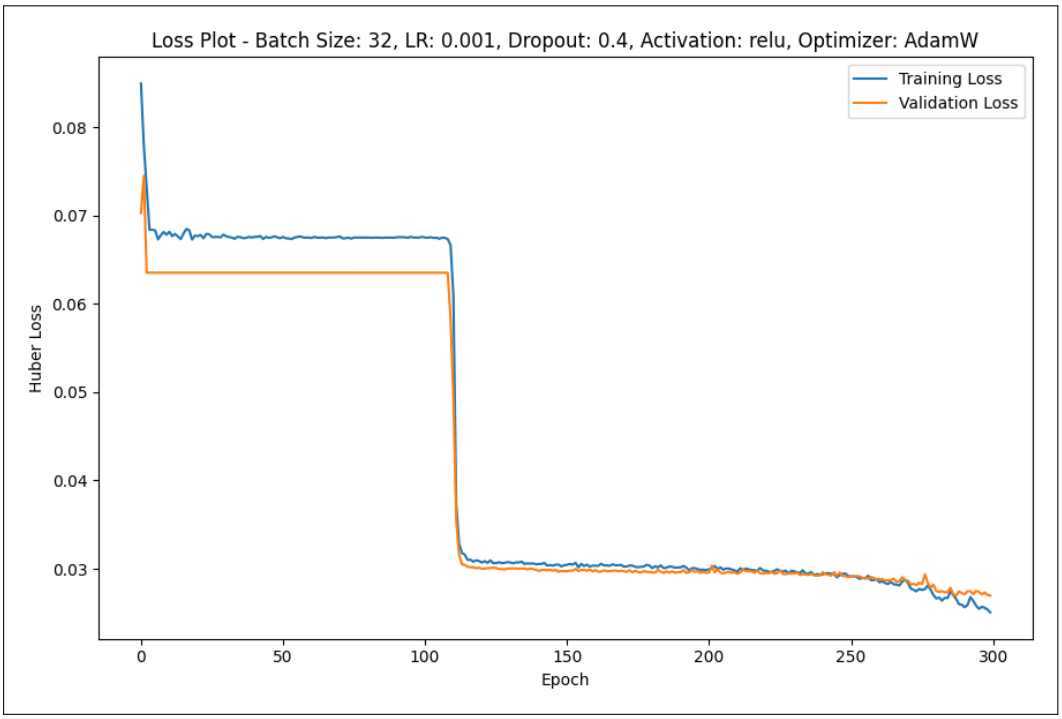

Figure J.1: Loss Plot for Model 4.

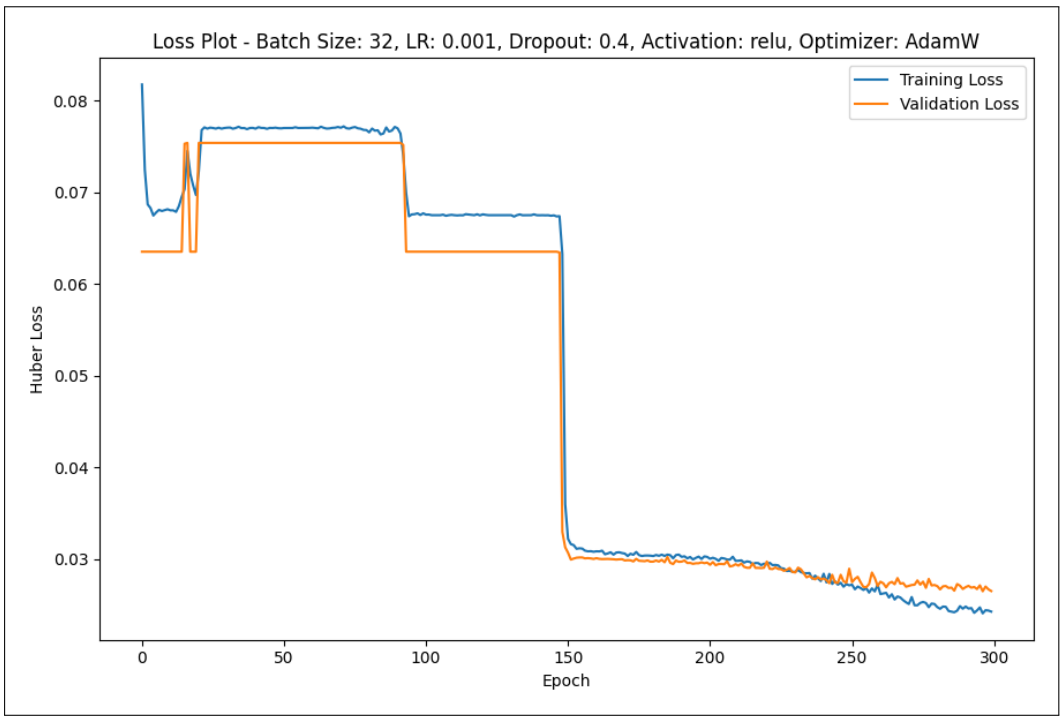

Figure J.2: Loss Plot for Model 5.

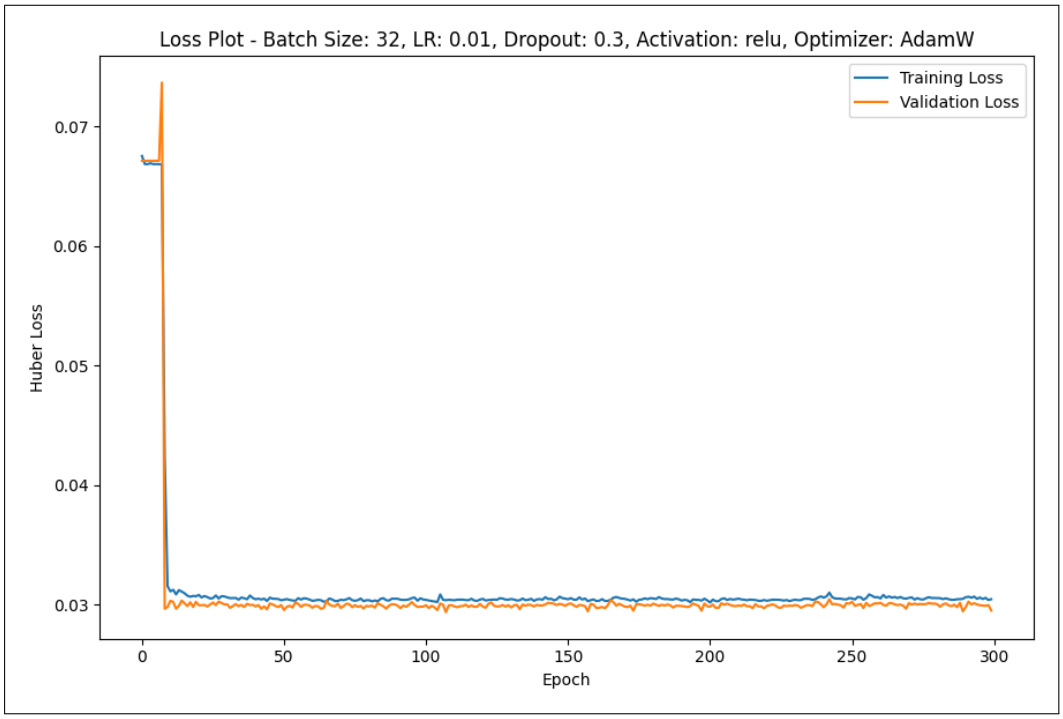

Figure J.3: Loss Plot for Model 6.

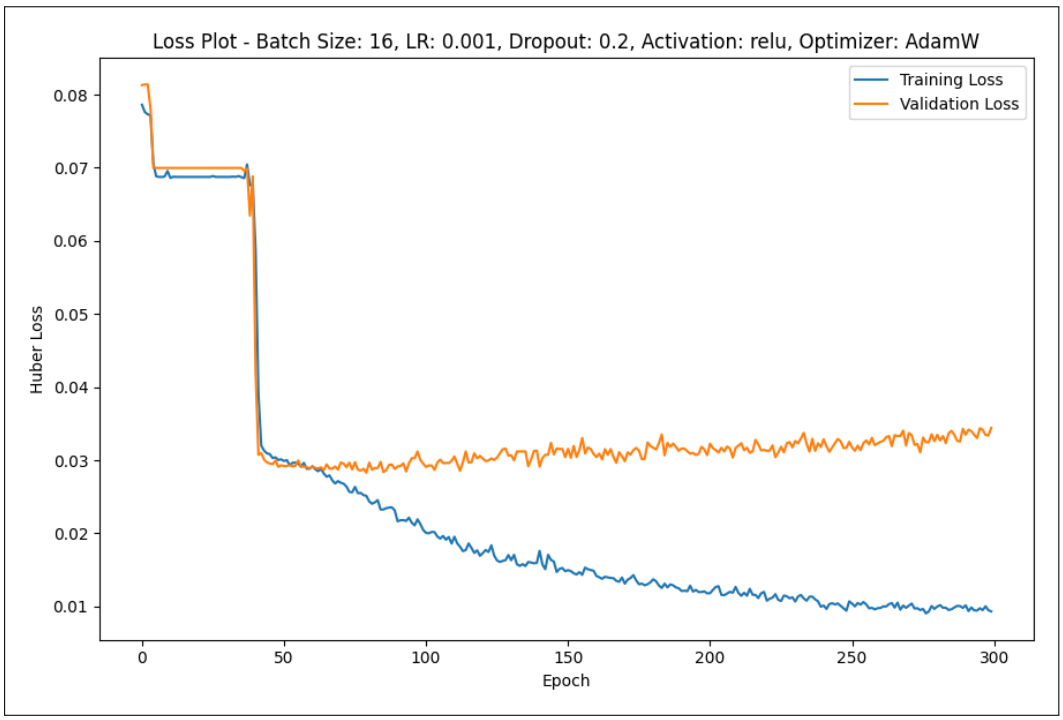

Figure J.4: Loss Plot for Model 12.

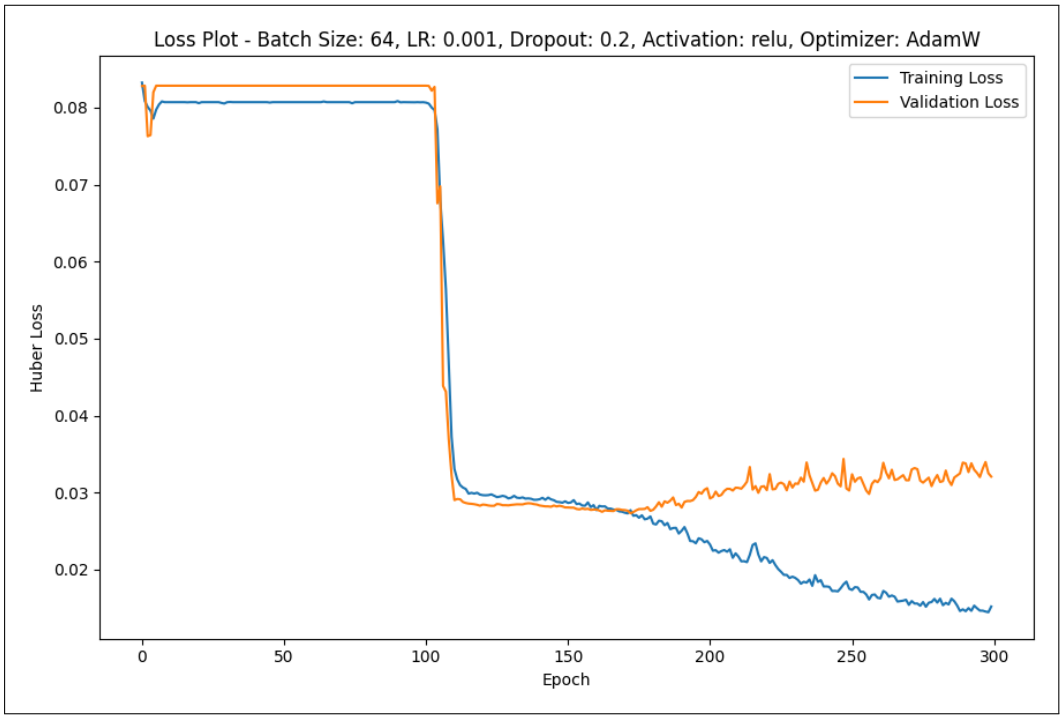

Figure J.5: Loss Plot for Model 13.

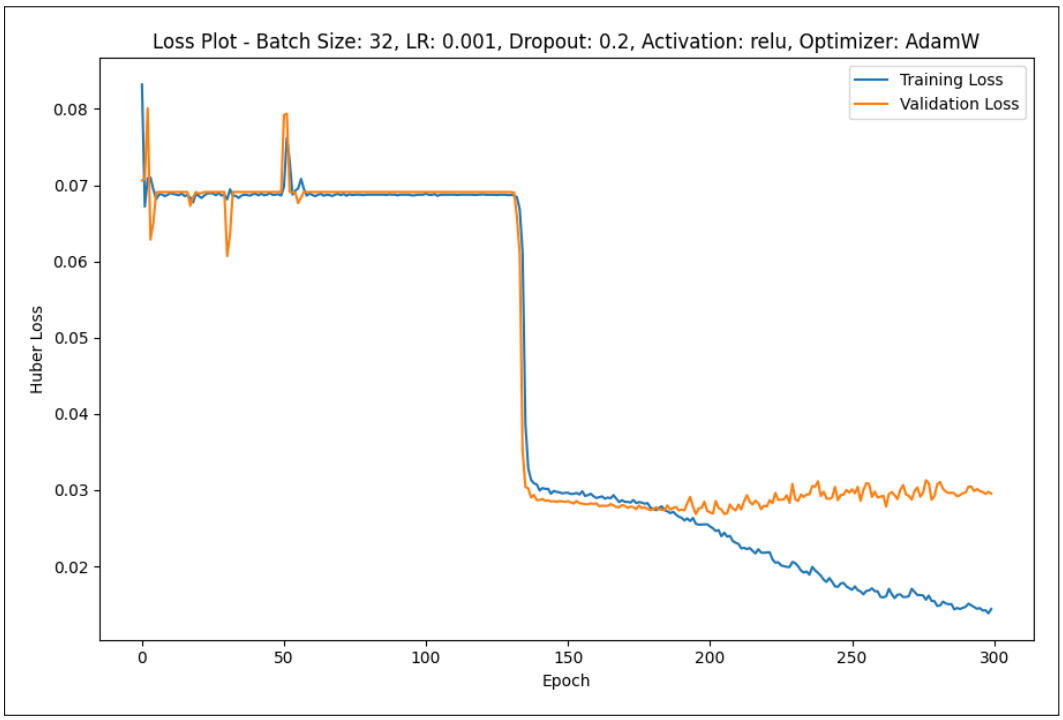

Figure J.6: Loss Plot for Model 14.

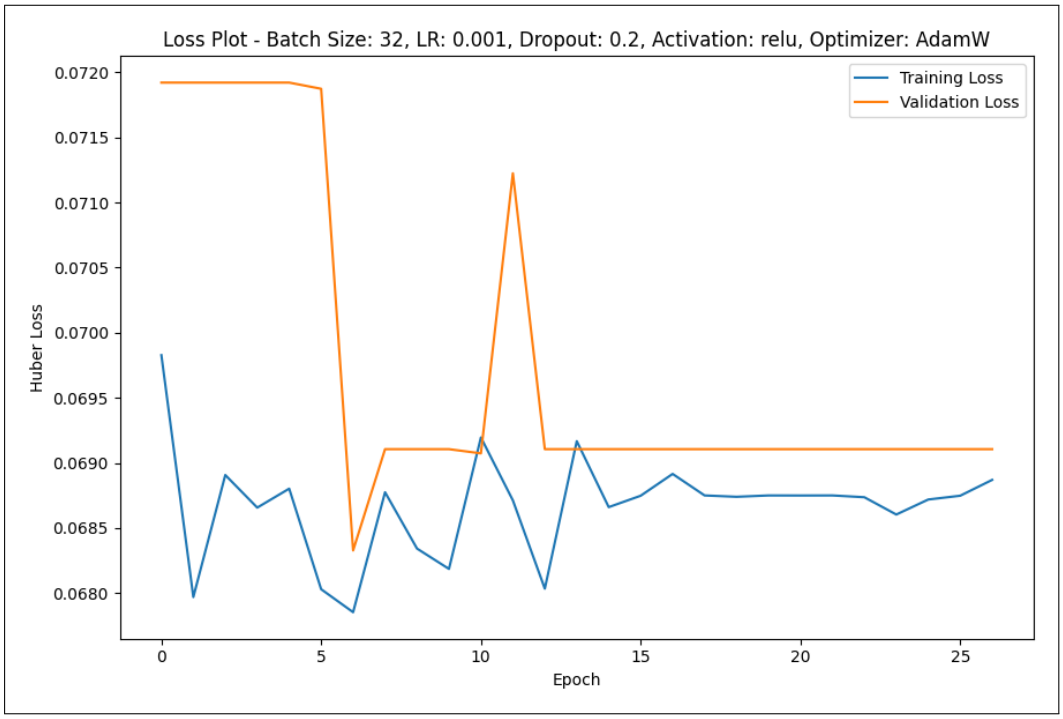

Figure J.7: Loss Plot for Model 15.

Predictive Plots

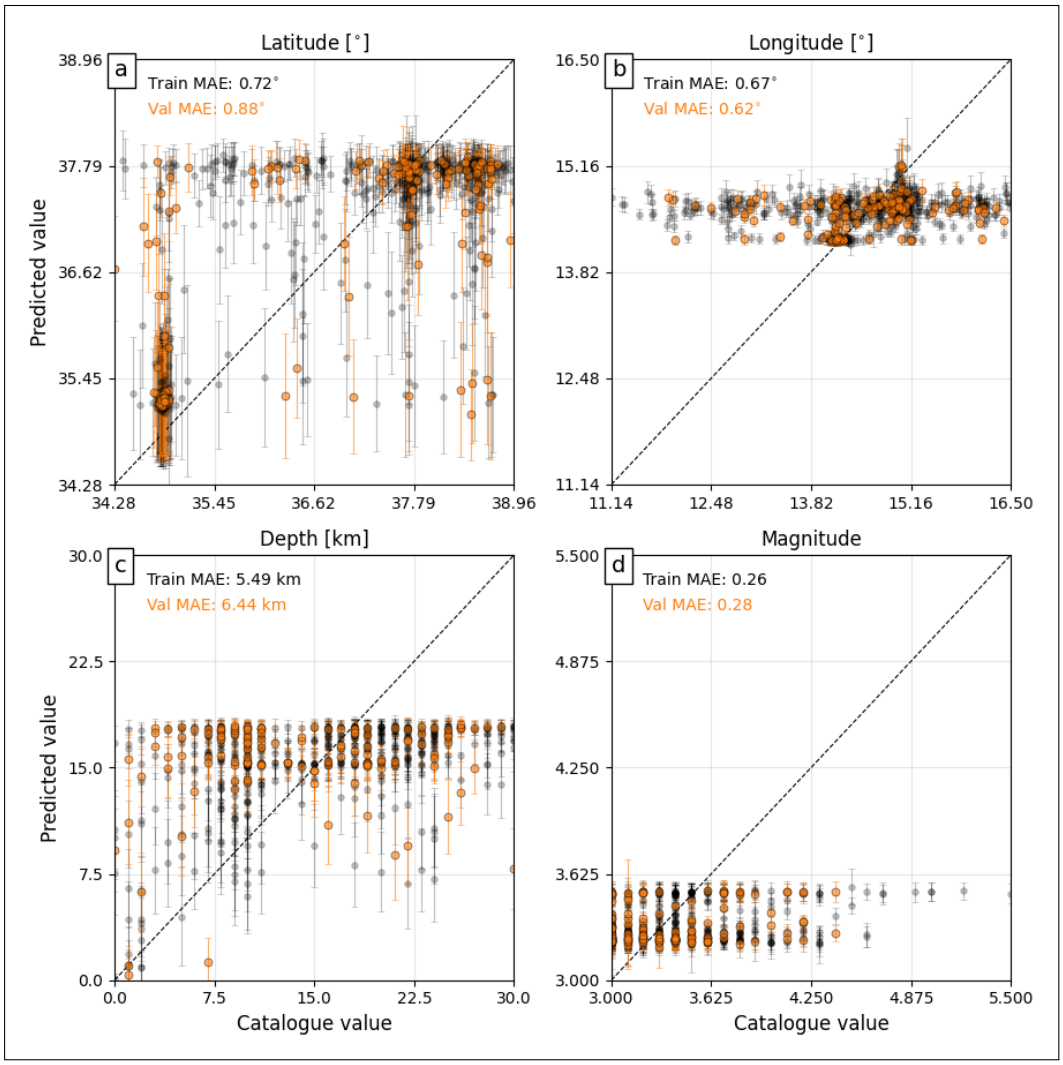

Figure J.8: Actual versus predicted target values for Model 4 with corresponding MAE computed on training and validation sets.

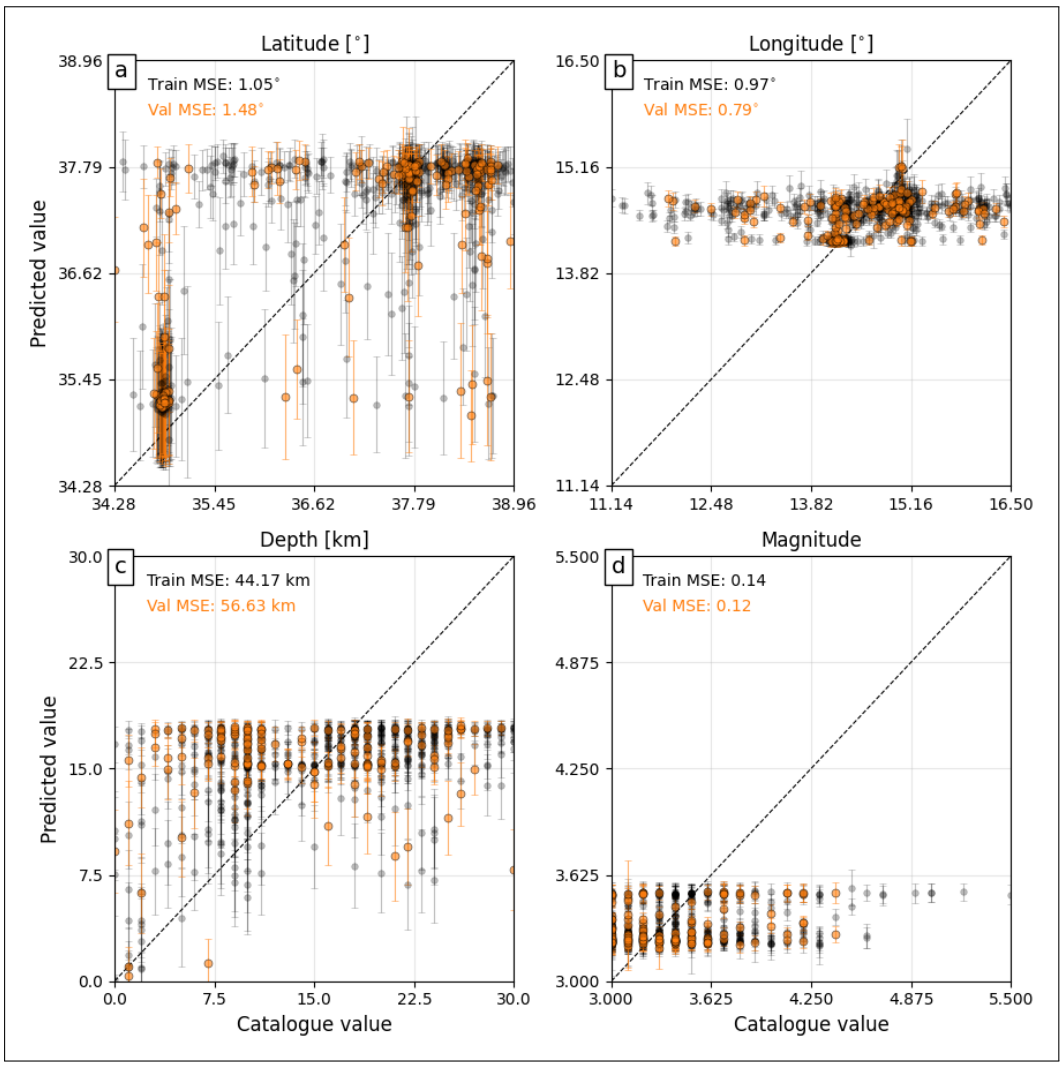

Figure J.9: Actual versus predicted target values for Model 4 with corresponding MSE computed on training and validation sets.

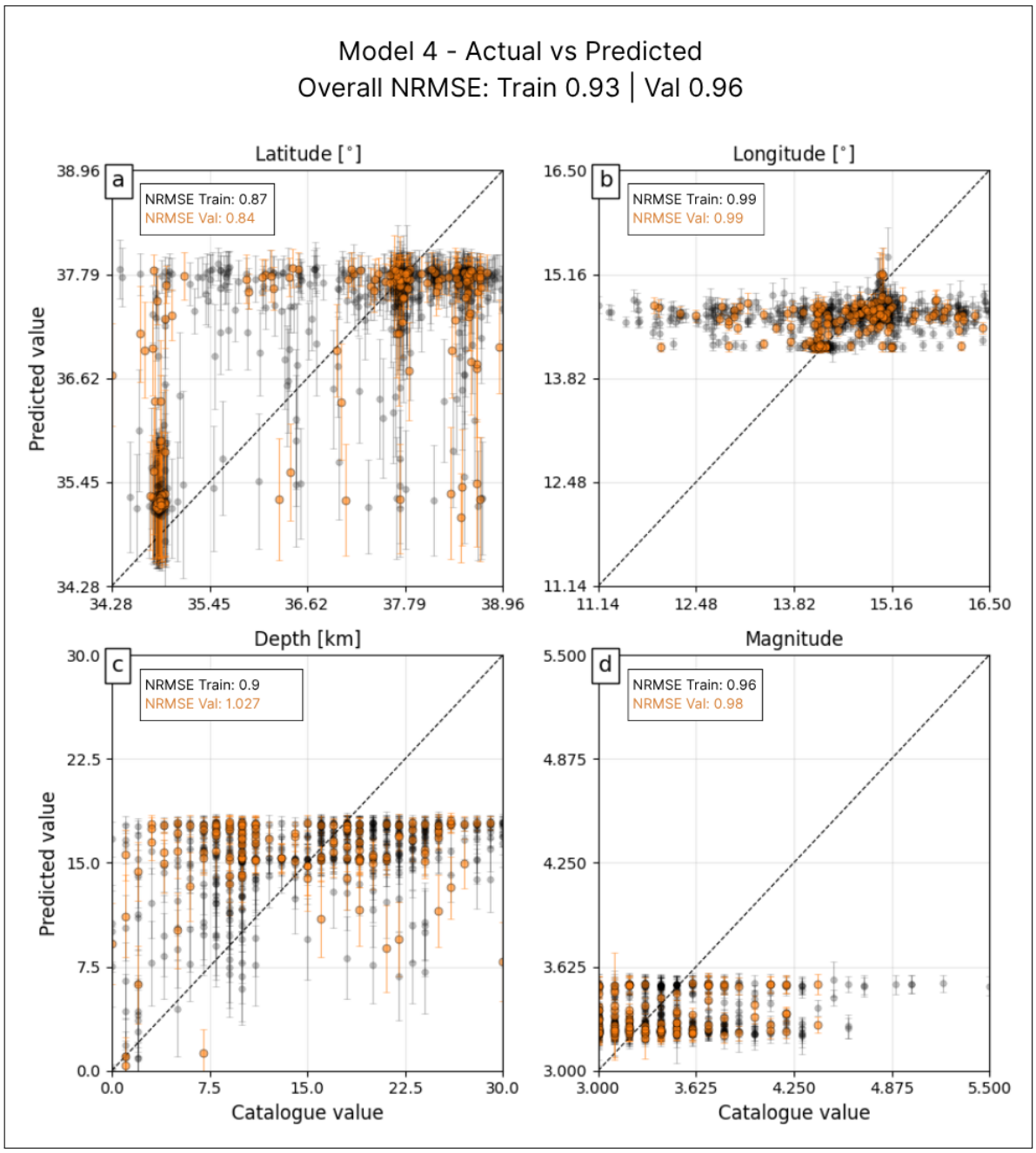

Figure J.10: Actual versus predicted target values for Model 4 with corresponding NRMSE computed on training and validation sets.

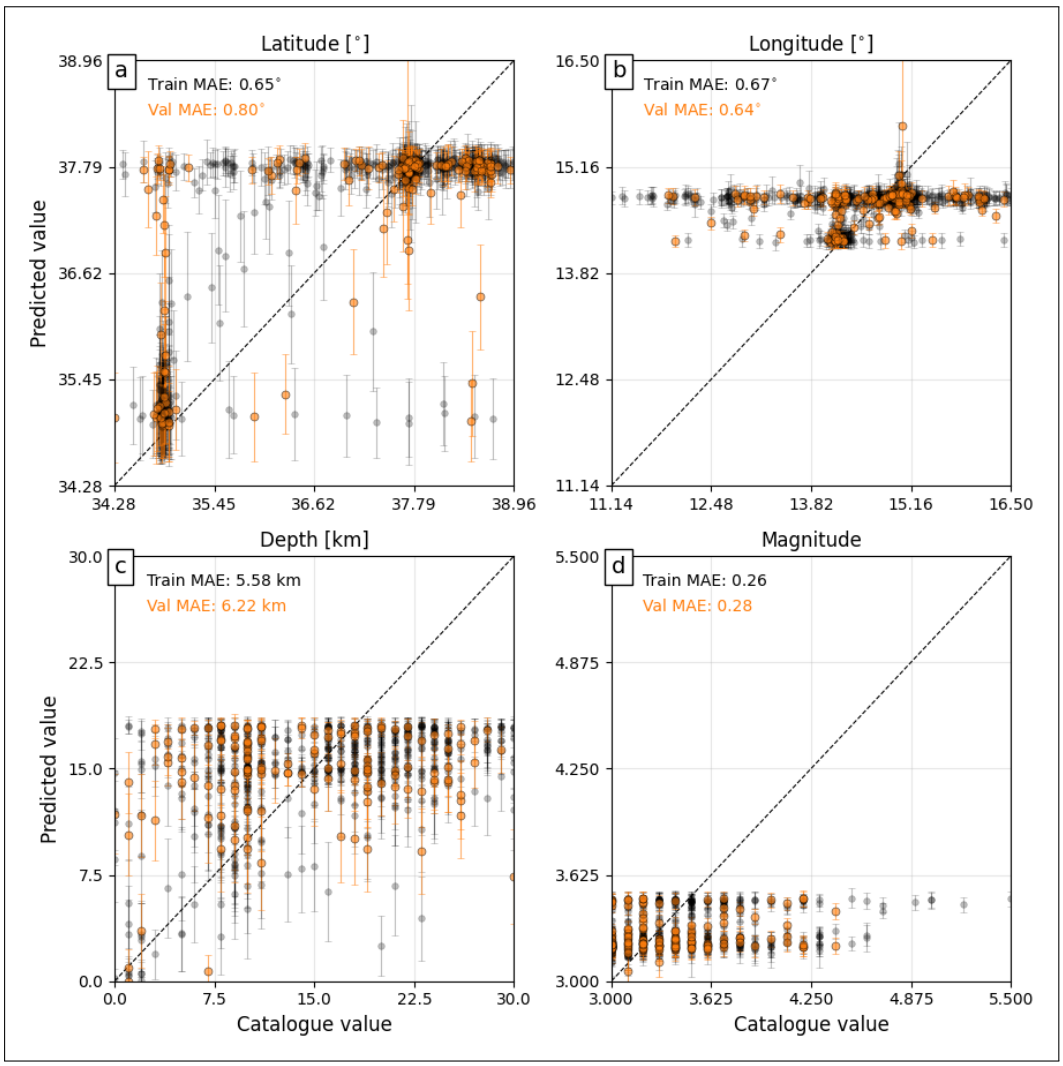

Figure J.11: Actual versus predicted target values for Model 5 with corresponding MAE computed on training and validation sets.

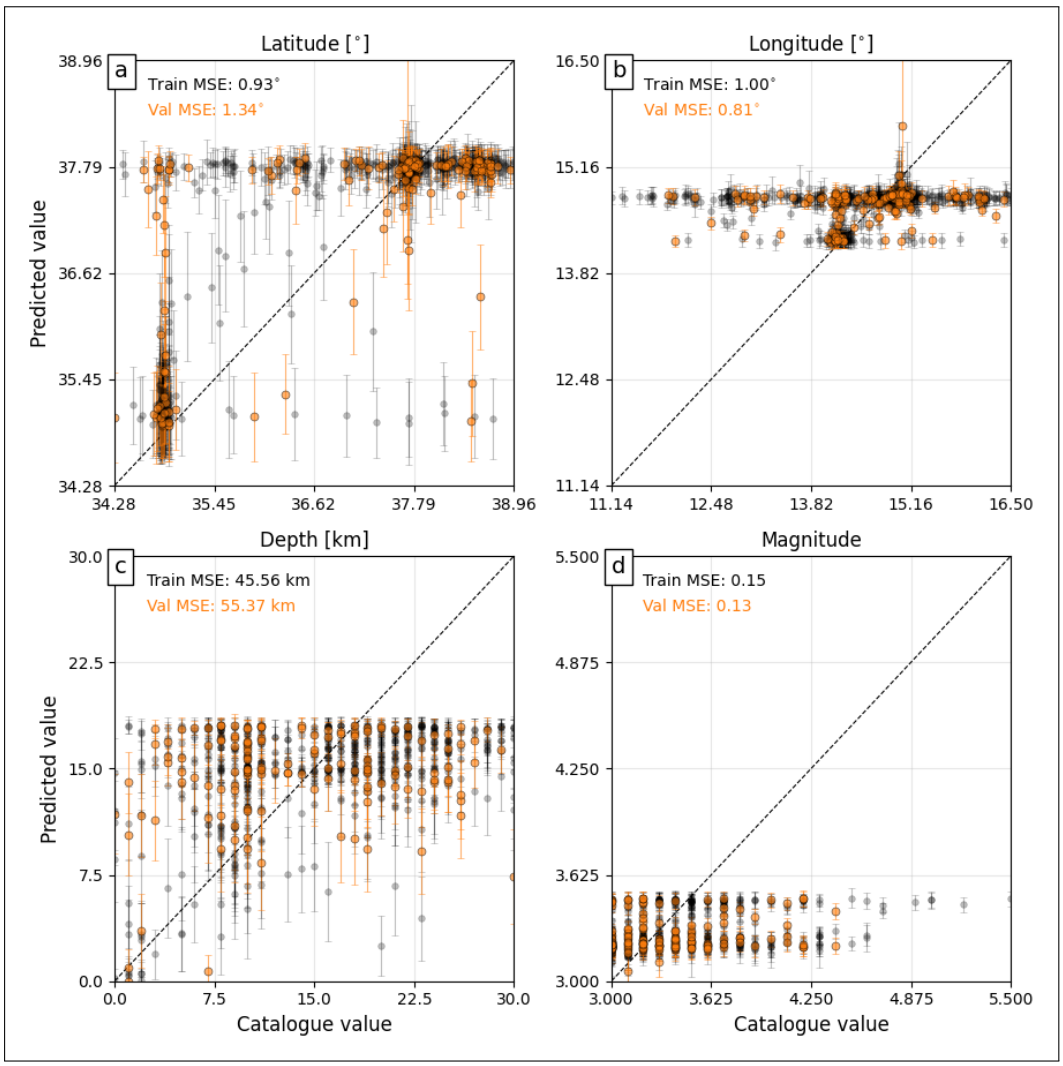

Figure J.12: Actual versus predicted target values for Model 5 with corresponding MSE computed on training and validation sets.

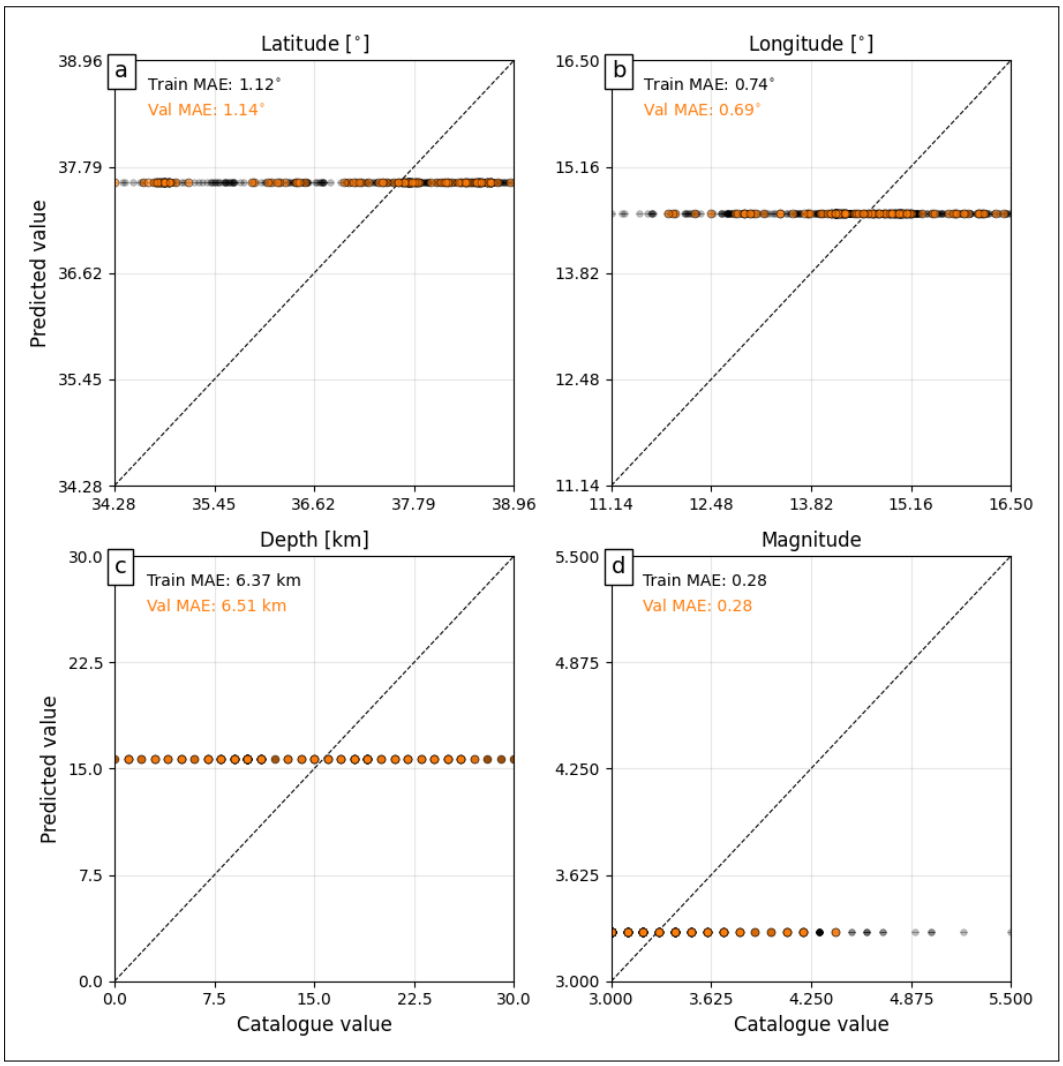

Figure J.13: Actual versus predicted target values for Model 6 with corresponding MAE computed on training and validation sets.

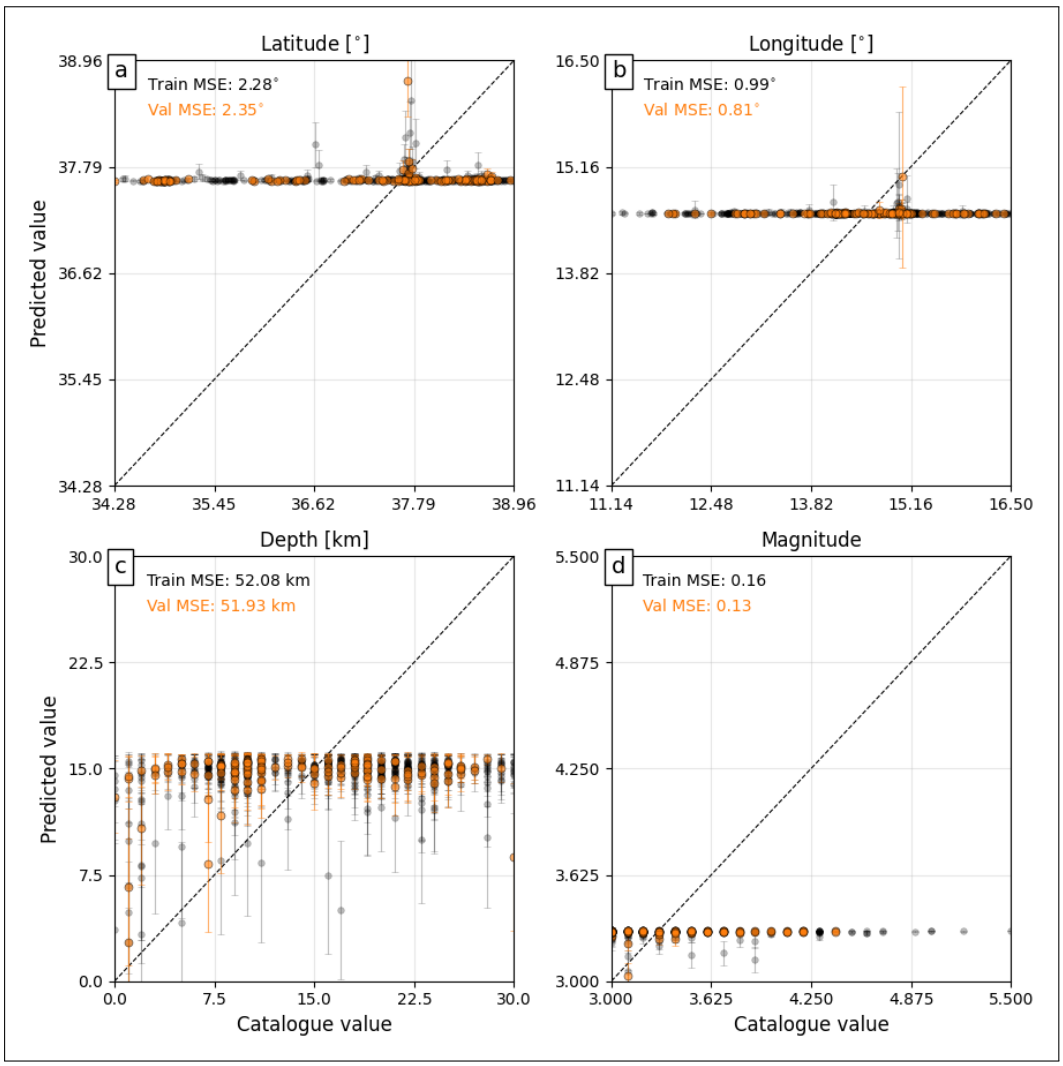

Figure J.14: Actual versus predicted target values for Model 6 with corresponding MSE computed on training and validation sets.

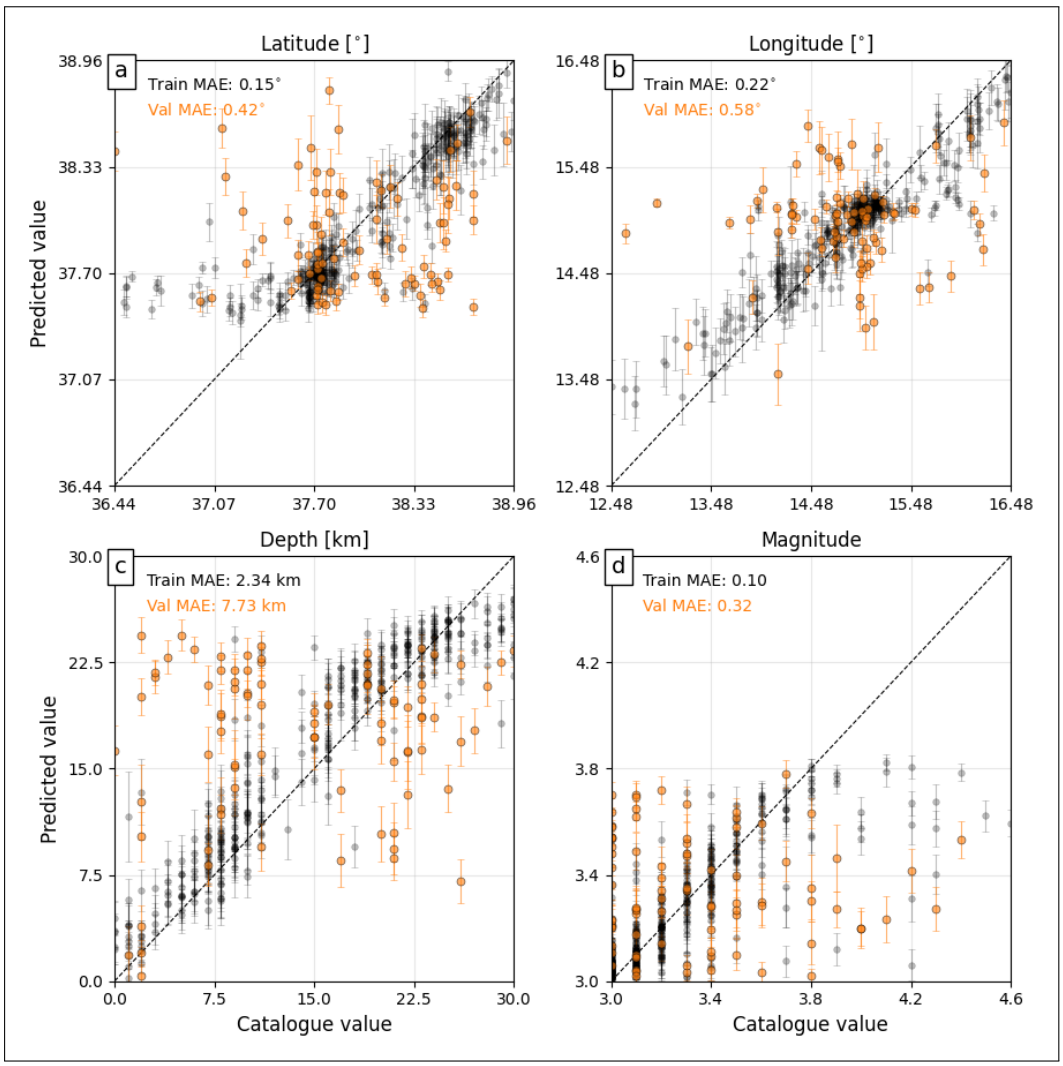

Figure J.15: Actual versus predicted target values for Model 12 with corresponding MAE computed on training and validation sets.

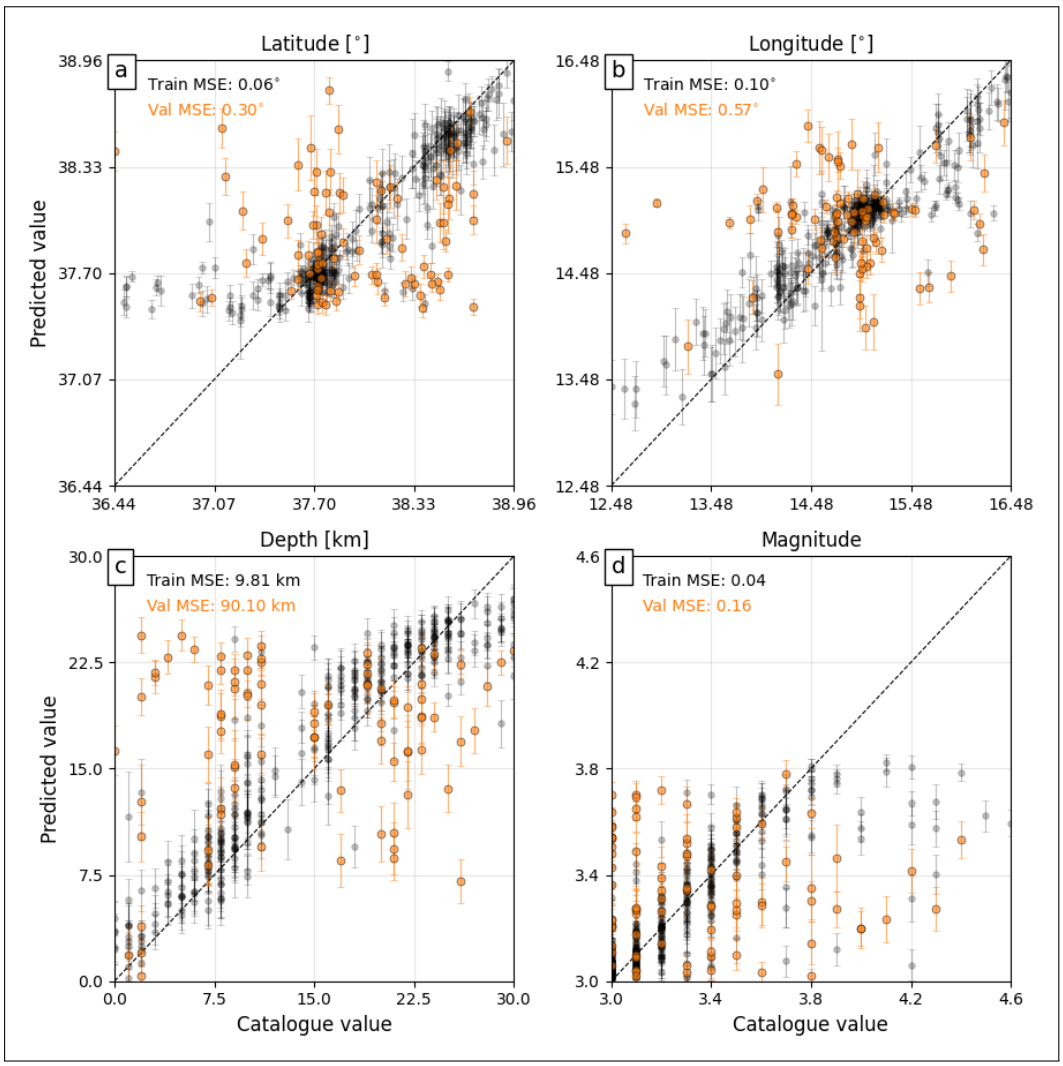

Figure J.16: Actual versus predicted target values for for Model 12 with corresponding MSE computed on training and validation sets.

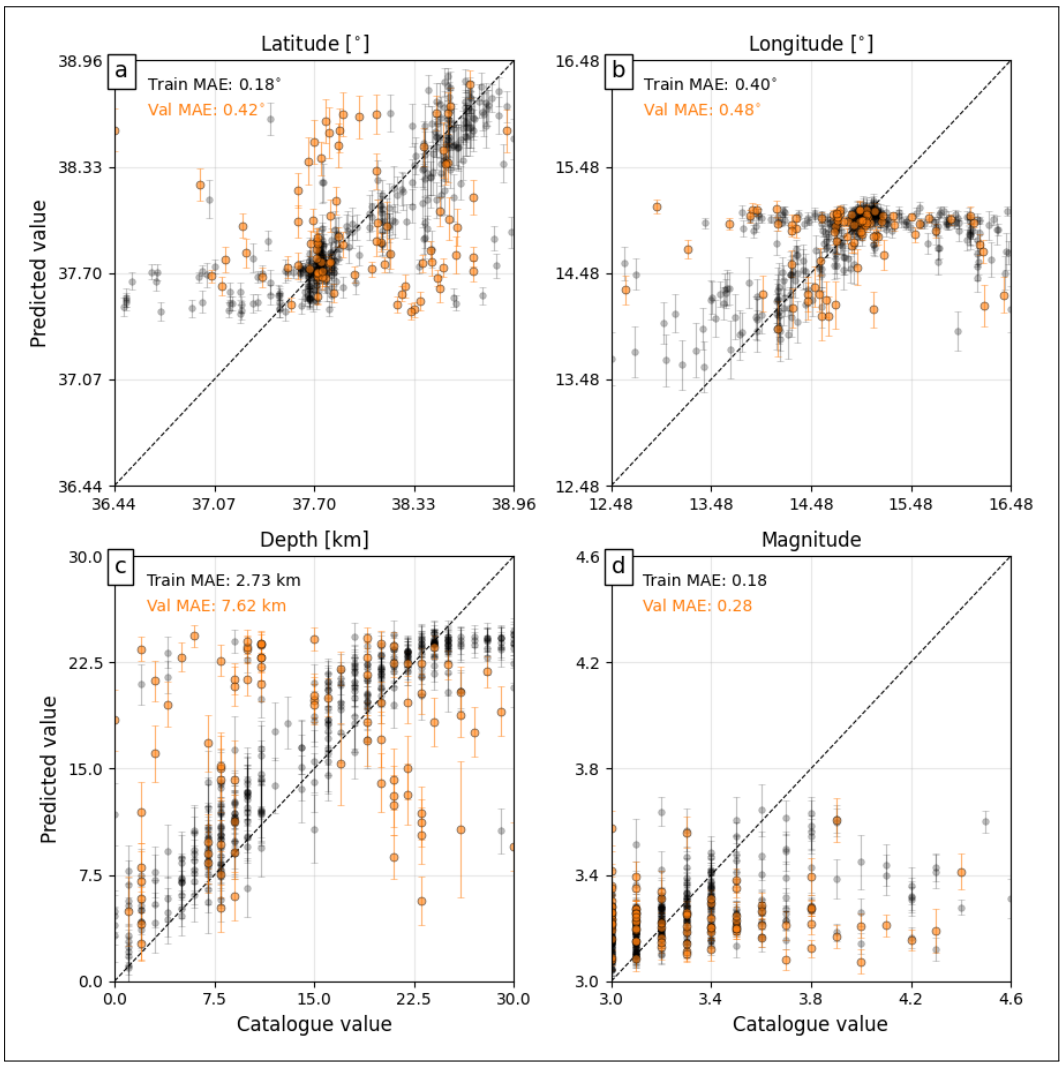

Figure J.17: Actual versus predicted target values for Model 13 with corresponding MAE computed on training and validation sets.

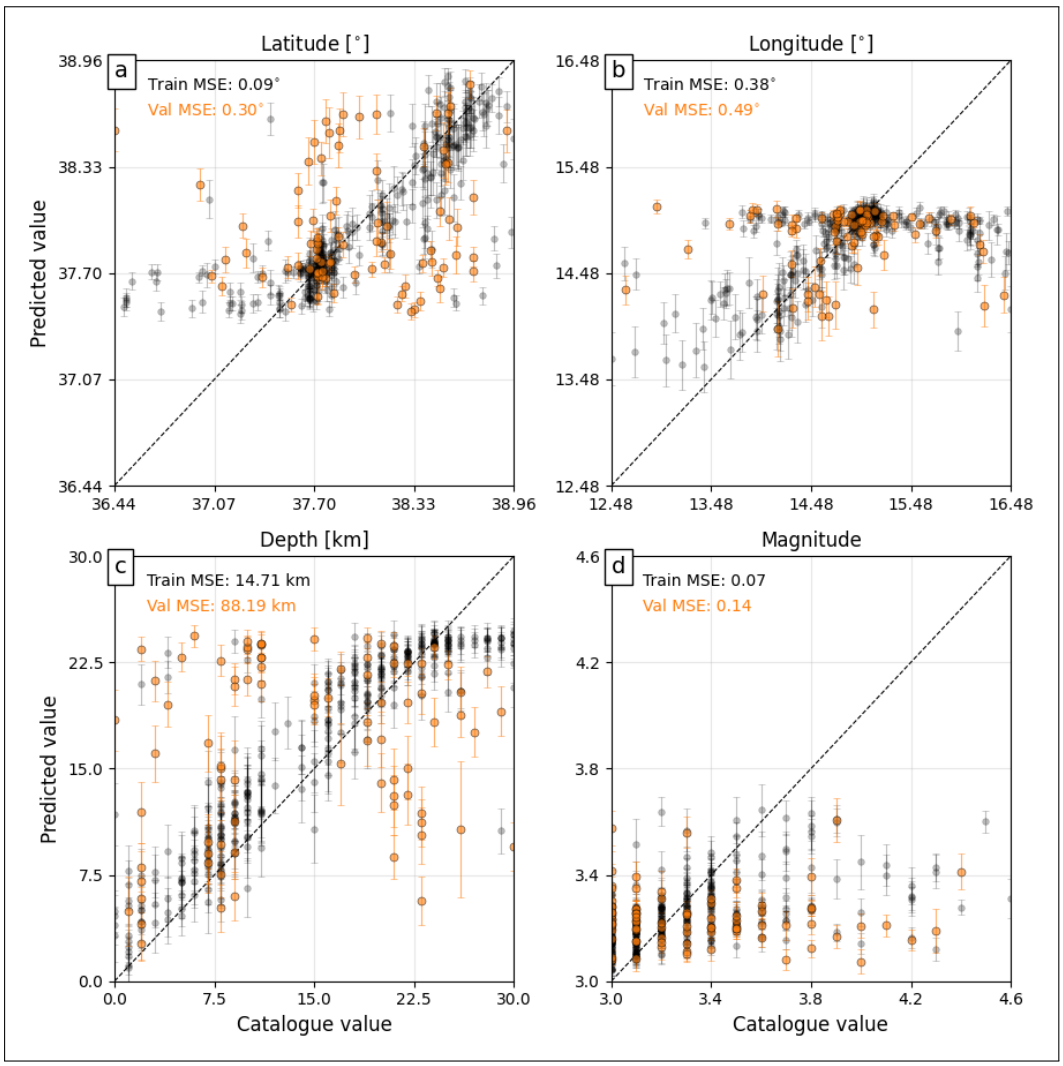

Figure J.18: Actual versus predicted target values for for Model 13 with corresponding MSE computed on training and validation sets.

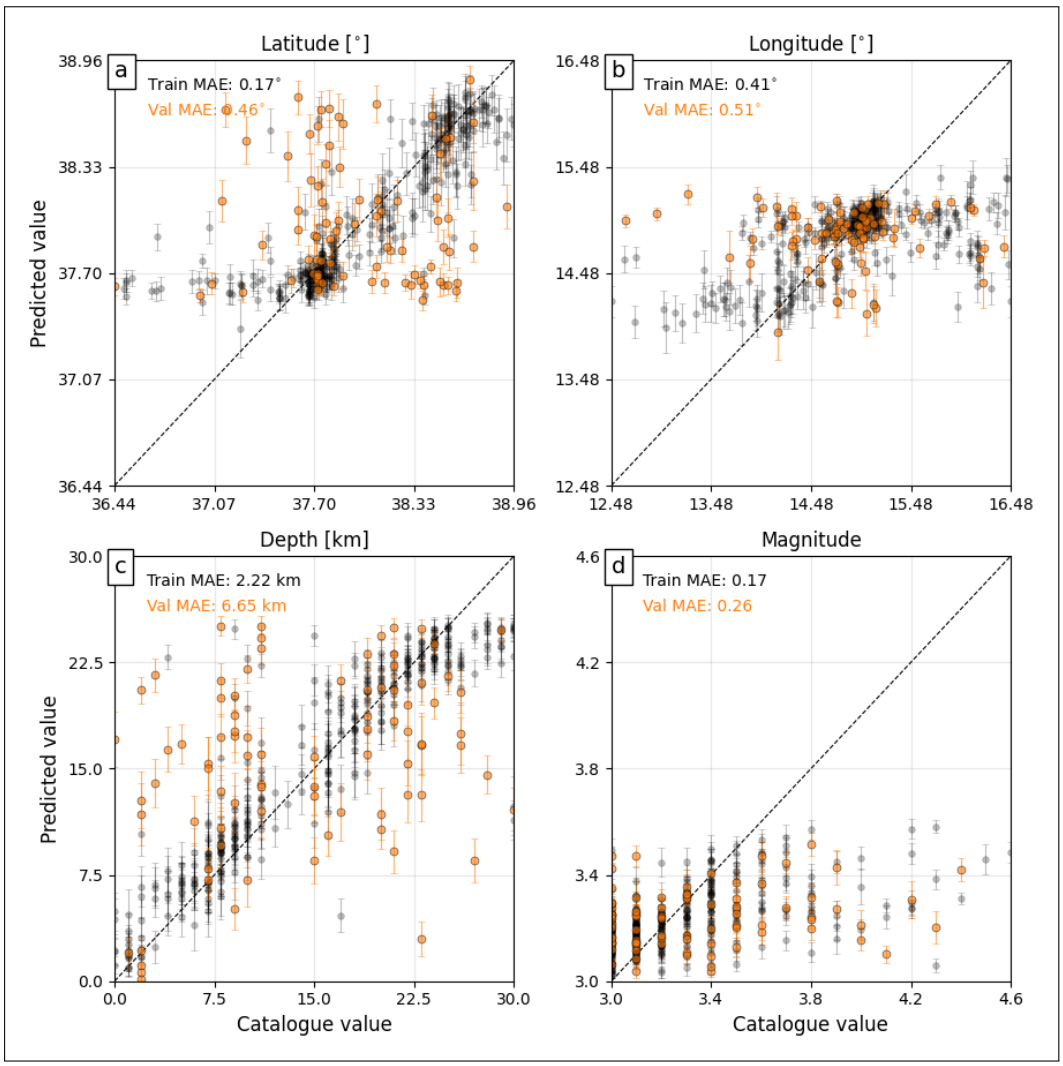

Figure J.19: Actual versus predicted target values for Model 14 with corresponding MAE computed on training and validation sets.

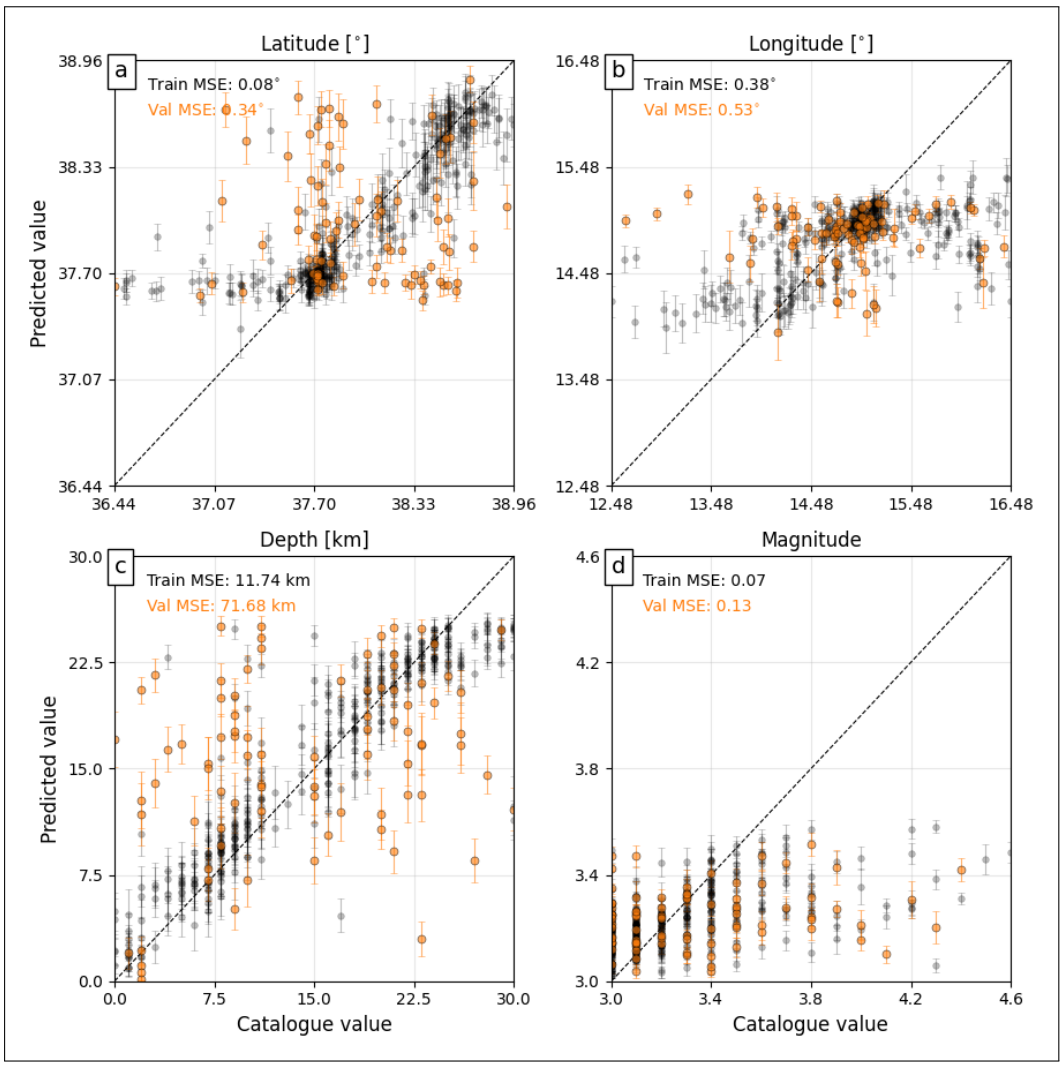

Figure J.20: Actual versus predicted target values for for Model 14 with corresponding MSE computed on training and validation sets.

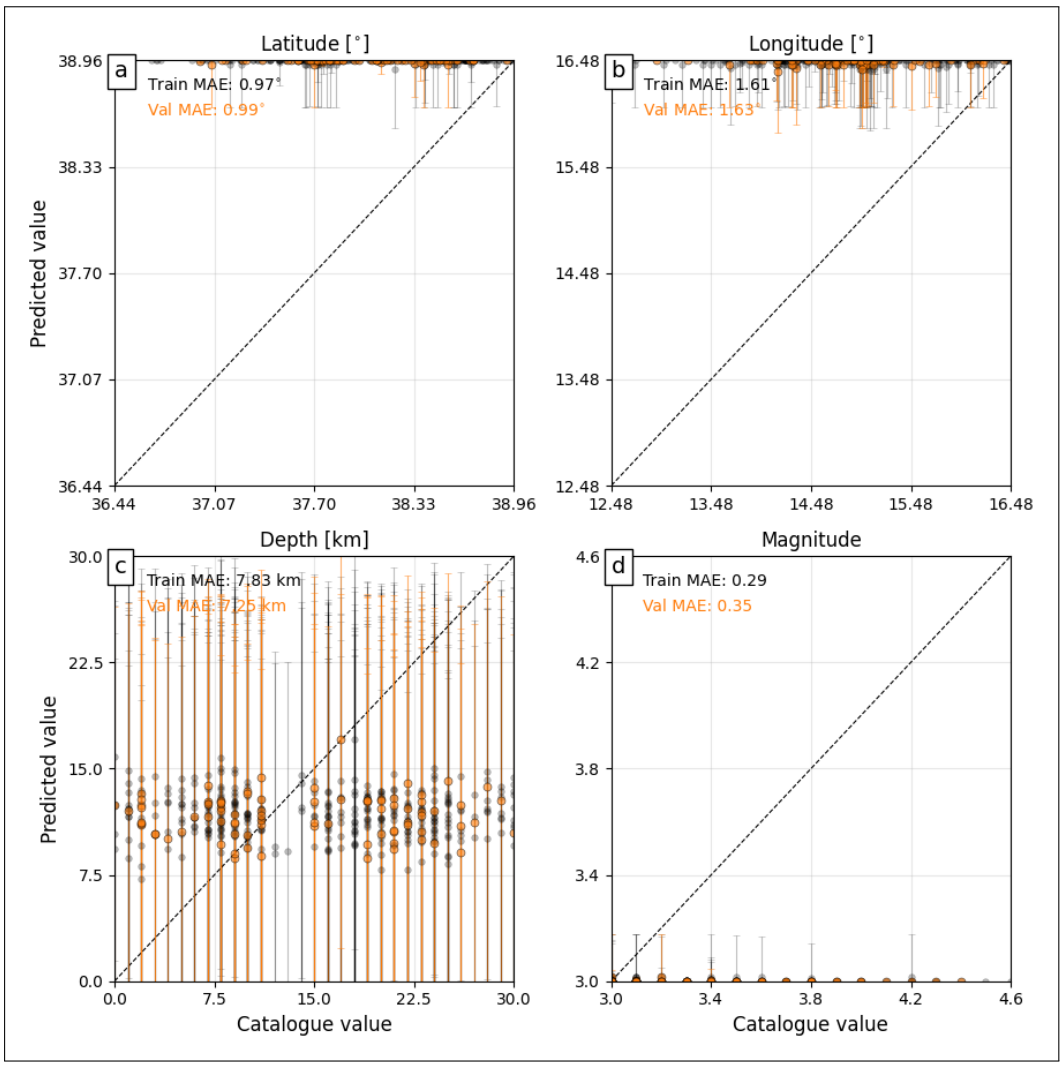

Figure J.21: Actual versus predicted target values for Model 15 with corresponding MAE computed on training and validation sets.

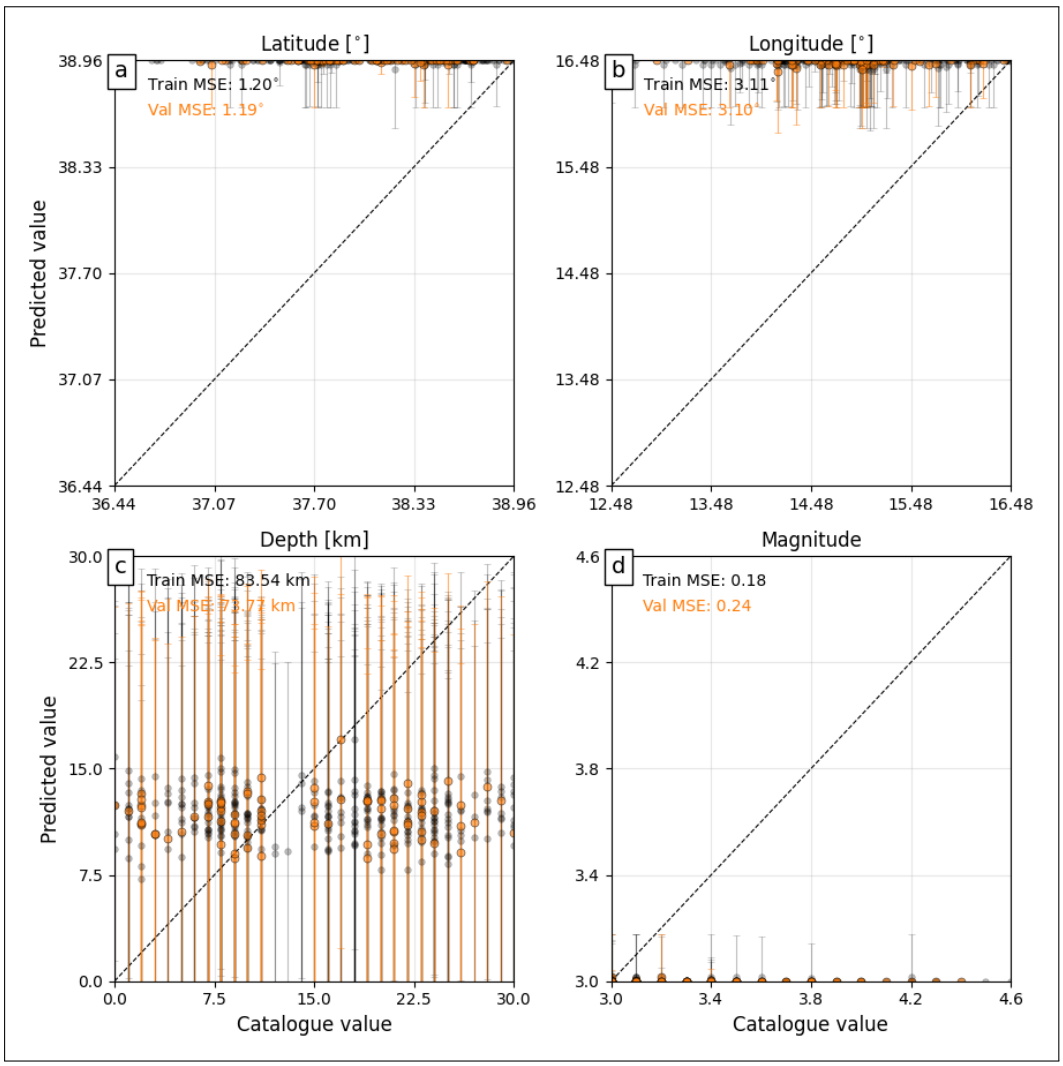

Figure J.22: Actual versus predicted target values for for Model 15 with corresponding MSE computed on training and validation sets.

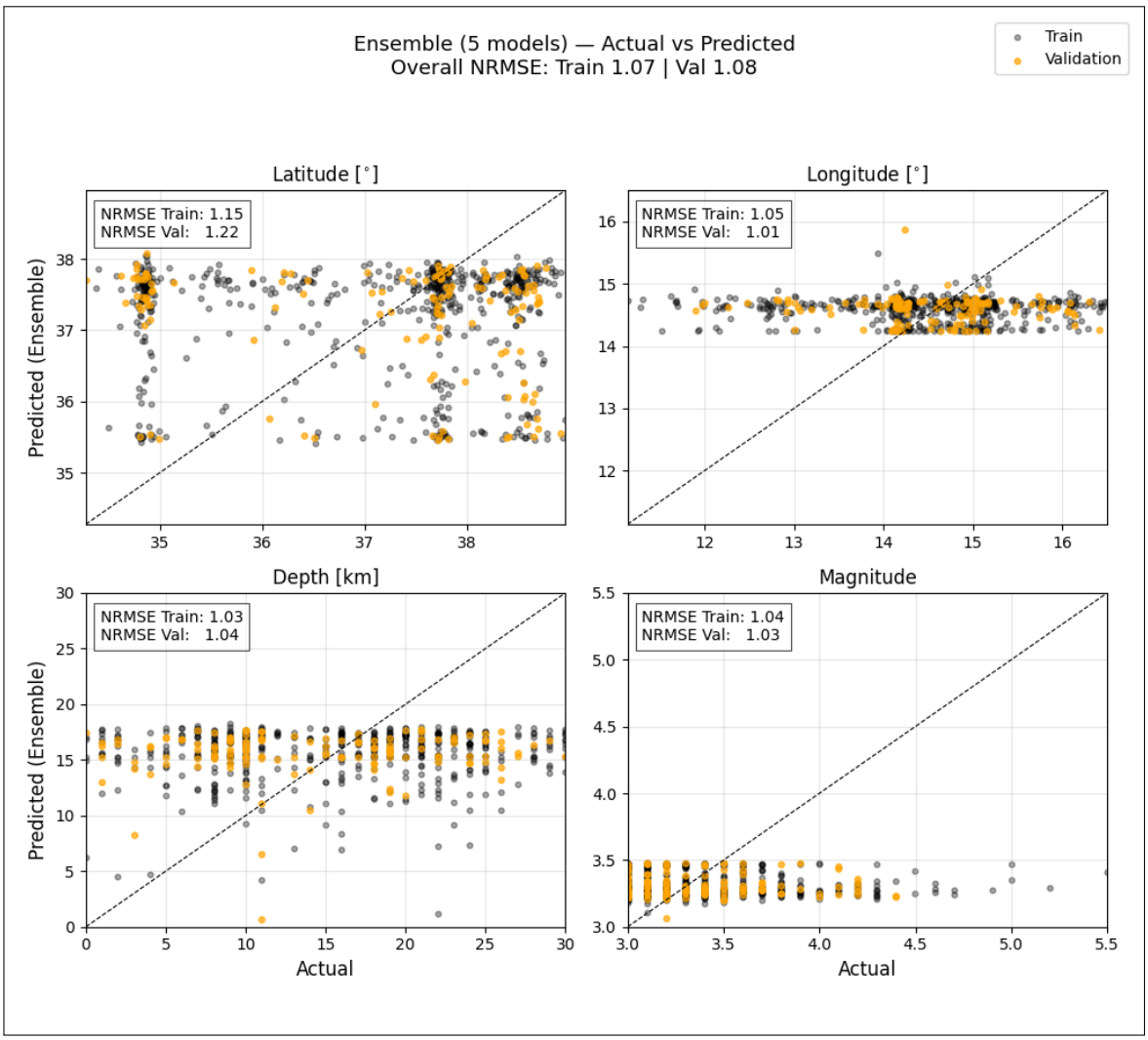

Figure J.23: Actual versus predicted target values for Dynamic Edges GNN Ensemble Model with corresponding NRMSE computed on training and validation sets.

Appendix K

Theory Related to Cluster Analysis

In this appendix, theory related to cluster analysis that is necessary for a complete understanding of the methods used in Section 5.7 is presented. The material presented here is meant solely to provide the mathematical background required for Section 5.7 and it should be noted that this notation is independent of, and should not be confused with, the notation used throughout the main body of the dissertation.

The term *cluster analysis* refers to a collection of statistical techniques that are employed to divide a population into K distinct but unknown groups, $C_1, ..., C_K$, referred to as clusters. The group membership of each vector of observations in the data is indicated by a latent random variable $\tilde{Y}$, conventionally referred to as the *target variable*. Since neither the number of clusters, K, nor the clusters themselves are known in advance, clustering methods operate in an unsupervised manner. Each element in the population is characterized by a set of attributes, modelled as random variables $X_1, ..., X_{n_x}$, which are assumed to follow a joint probability distribution $f(x_1, ..., x_{n_x}, \tilde{y})$. Based on this distribution, clustering also seeks to identify underlying groupings by exploiting patterns in the data, forming clusters such that elements within the same cluster are similar to each other, while those in different clusters are dissimilar.

There are various types of clustering techniques which can be categorized into *partitioning methods*, *hierarchical methods*, *model-based methods*, *density-based methods* and *grid-based methods*. This dissertation uses K-means, DBSCAN and HDBSCAN. K-means is a partitioning method which will be outlined in more detail in Section K.1.1. DBSCAN is a density-based method and will be discussed in Section K.2. HDBSCAN extends DBSCAN by incorporating a hierarchical approach and will be presented in Section K.3.

K.1 Partitioning Methods and K-means Clustering

Given vectors of observations $\boldsymbol{x}_1, ..., \boldsymbol{x}_N$ where each vector is n_x-dimensional, partitioning clustering methods aim to divide the N elements into $K \leq N$ clusters. The number of clusters, K, is assumed to be known, and needs to be pre-specified by the user. There are several methods within the literature, such as the *elbow method*, *silhouette method* and *gap statistic*, to determine K. More detail on these methods is provided in Section K.1.2. In partitioning methods, each cluster must contain at least one element and each element must belong to exactly one cluster. This is termed *exclusive cluster separation*.

The most well-known partitioning method is the K-means method. Its roots can be traced back to Steinhaus (1957), however, the term was first coined in MacQueen (1967). K-means clustering assigns each element to the cluster having the nearest centroid or mean. The process will now be detailed.

K.1.1 K-means

The K-means algorithm begins by selecting g elements to serve as *'seeds'*. These 'seeds' may be selected using different approaches. One approach is to randomly choose the g elements without imposing any constraints. Another method is to select the g elements based on a distance criterion. Specifically, that the elements selected are at least d units of distance apart from each other. The most common distance metric used is the *Euclidean distance* (Leskovec et al., 2014). Given two data points $\boldsymbol{x}_i = (x_{i1}, ..., x_{in_x})'$ and $\boldsymbol{x}_{i'} = (x_{i'1}, ..., x_{i'n_x})'$, $i, i' = 1, ..., N$, $i \neq i'$, the Euclidean distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i'}$ is calculated using the following formula:

$$d_{EUC}(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) = \sqrt{\sum_{j=1}^{n_x}(x_{ij} - x_{i'j})^2}. \tag{K.1}$$

Other distance metrics found within the literature include the *Manhattan distance* (Leskovec et al., 2014), the *Minkowski distance* (Leskovec et al., 2014), *Jaccard distance* (dissimilarity measure) (Leskovec et al., 2014), *Gower's distance* (Gower, 1971) and distance metrics based on correlation measures.

After each of the g initial elements have been selected, each of the $N-g$ remaining observations is assigned to the nearest 'seed' according to some distance metric. As soon as the cluster has more than one element, the 'seed' is replaced with the sample mean vector. Once all elements are assigned to their clusters, thus creating an *initial*

partitioning, an iterative technique is employed. This technique tries to improve the initial partitioning by transferring elements from one cluster to another. This is carried out by first computing and comparing the distance between each element and the centroids of all the clusters. If the smallest distance is between the element and a centroid from another cluster, the element is reallocated to that cluster. When a cluster gains or loses an element, the algorithm recomputes its centroid, and the distances between every set of elements and the updated centroids are calculated once again. This procedure is repeated until elements are no longer reallocated. A 'good' clustering is obtained when the sum of squared distances between all elements in a cluster and their centroid is minimized as much as possible.

It is worth noting that since the K-means method is sensitive to the choice of the initial 'seeds' (Blashfield, 1976; Friedman & Rubin, 1967), it is recommended to repeat the clustering procedure more than once using different 'seeds'.

K.1.2 Determining the Number of Clusters

Deciding on the appropriate number of clusters is challenging since there is no known optimal amount. Nevertheless, numerous methods can be used to determine the number of clusters, however, in this dissertation the *elbow method*, *the gap statistic* and the *silhouette score* were used.

The elbow method (Thorndike, 1953) is a visual approach for determining the optimal number of clusters in a dataset. It is based on the principle that increasing the number of clusters, reduces the Within Cluster Variation (WCV). However, beyond a certain point, adding more cluster yields diminishing returns in reducing the WCV, resulting in a notable change in the slope of the WCV curve. This point, resembling an "elbow", is considered the "optimal" number of clusters. The elbow method is performed as follows:

1. Given a number, $K > 0$, and a dataset $\mathbb{X}$, K clusters are formed using a clustering algorithm such as K-means.
2. For each K value, the WCV is computed.
3. The curve of WCV against K is plotted.
4. The first or more significant turning point of the curve, i.e. the elbow, then indicates the "optimal" number of clusters.

The gap statistic (Tibshirani et al., 2002) determines the "optimal" number of

clusters by testing the observed clustering performance against expected clustering performance under a null reference model. In particular, it measures the difference (the "gap") between the total WCV for a given number of clusters K and its expected value under the null reference distribution. The K that outputs the largest gap statistic is then chosen to be the ideal amount of clusters. This suggests that the data clustering structure is significantly different from a random uniform distribution of points. In other words, the bigger the gap statistic, the more meaningful are the clusters identified. The algorithm is carried out in the following way:

1. Cluster analysis is performed on the observed data, varying the number of clusters $K > 0$.
2. For each $K > 0$, the WCV is computed on the observed data.
3. B reference datasets with a random uniform distribution are generated. Cluster analysis is performed on each of the B reference datasets, varying the number of clusters $K > 0$.
4. For each $K > 0$, the WCV is computed on the B reference datasets.
5. The estimated gap statistic is computed as the distance between the observed WCV_K and its expected value WCV_{KB} under the null hypothesis:

$$Gap(K) = \frac{1}{B}\sum_{b=1}^{B} log(WCV_{KB}) - log(WCV_K)$$

6. The standard deviation of the gap statistic is also computed.
7. The smallest value of K such that the gap statistic is within one standard deviation of the gap at $K+1$, i.e. $Gap(K) \geq Gap(K+1) - s_{K+1}$, is chosen to be the optimal number of clusters.

The silhouette score (Rousseeuw, 1987) is an internal cluster validation metric used to assess how well data points have been assigned to clusters, however, it can also be used to determined the optimal number of clusters. Suppose D is the set of all observations from $\mathbb{X}$, i.e. $D = \{\boldsymbol{x}_1, \ldots \boldsymbol{x}_N\}$. Let D, be partitioned into K clusters, $C_1, ..., C_K$. For every observation $\boldsymbol{x}_i \in D$, $i = 1, ..., N$ the average distance between $\boldsymbol{x}_i$ and all other elements in the cluster to which $\boldsymbol{x}_i$ is a part of is calculated and denoted as $a(\boldsymbol{x}_i)$. Additionally, the minimum average distance between $\boldsymbol{x}_i$ and all the clusters in which $\boldsymbol{x}_i$ does not belong, is calculated and denoted as $b(\boldsymbol{x}_i)$. Thus, for $\boldsymbol{x}_i \in C_k$, $k = 1, ..., K$, we have:

$$a(\boldsymbol{x}_i) = \frac{\sum_{\boldsymbol{x}_{i'} \in C_k, \boldsymbol{x}_i \neq \boldsymbol{x}_{i'}} dist(\boldsymbol{x}_i, \boldsymbol{x}_{i'})}{|C_k| - 1}, \tag{K.2}$$

and

$$b(\boldsymbol{x}_i) = \min_{C_{k'}: 1 \leq k' \leq K, k' \neq k} \left\{ \frac{\sum_{\boldsymbol{x}_{i'} \in C_{k'}} dist(\boldsymbol{x}_i, \boldsymbol{x}_{i'})}{|C_{k'}|} \right\}. \tag{K.3}$$

The silhouette coefficient of $\boldsymbol{x}_i$ can then be defined as:

$$s(\boldsymbol{x}_i) = \frac{b(\boldsymbol{x}_i) - a(\boldsymbol{x}_i)}{\max\{a(\boldsymbol{x}_i), b(\boldsymbol{x}_i)\}}. \tag{K.4}$$

The silhouette coefficient lies within the range $(-1, 1)$. The value of $a(\boldsymbol{x}_i)$ indicates how compact the cluster which $\boldsymbol{x}_i$ forms part of is. The smaller $a(\boldsymbol{x}_i)$ is, the more compact the cluster. On the other hand, the value of $b(\boldsymbol{x}_i)$ measures how separated $\boldsymbol{x}_i$ is from other clusters. The larger the value of $b(\boldsymbol{x}_i)$, the more separated $\boldsymbol{x}_i$ is from other clusters. Thus, when $s(\boldsymbol{x}_i) \to 1$, the cluster containing $\boldsymbol{x}_i$ is compact and $\boldsymbol{x}_i$ distant from the other clusters, which is what is preferred. On the contrary, a negative silhouette coefficient indicates that in expectation, $\boldsymbol{x}_i$ is closer to the elements in another cluster than to the elements in the same cluster as $\boldsymbol{x}_i$. This situation is inadequate and should be avoided.

The average silhouette coefficient value of all the elements in the cluster can be used to determine how well a cluster fits within the resulting cluster solution. Additionally, the average silhouette coefficient value of all the elements in the dataset can be used to measure the clustering quality. An average silhouette score of 1 indicates that the clusters are very dense and nicely separated. A score of 0 means that clusters are overlapping. A score of less than 0 means that elements belonging to clusters may be wrong/incorrect. The silhouette coefficient and other intrinsic methods can also be utilized in the elbow method to heuristically obtain the optimal number of clusters in a dataset by substituting them instead of the sum within-cluster variances. The optimal number of clusters is typically chosen as the value of K that maximizes the silhouette score.

K.2 Density-Based Methods and DBSCAN

Prior to discussing DBSCAN it is important to distinguish between the two different types of density present within the clustering literature. *Data density* refers to how

densely packed observations are in certain regions of the data space, without assuming an underlying probability distribution. In this case, the density of a region is determined by counting the number of observations within it, and a region is considered dense if the number of observations exceeds a pre-specified density threshold. Conversely, *probability density* is related to the probability density function in the usual statistical sense. In this case, clusters are seen as regions in the data space where the probability density function is high. This type of density will not be considered here. As a result the term *density* will refer to data density from here onwards.

In DBSCAN, the density around a data point $\boldsymbol{x}_i \in \mathbb{R}^{n_x}$, $i = 1, ..., N$, is measured by defining a region around said point, and examining the number of data points within the designated region. These data points are referred to as neighbours. The size of the region is determined by a hyperparameter denoted as Eps, which represents the maximum distance between two points for them to be considered neighbours. In addition to Eps, the algorithm uses another hyperparameter known as $MinPts$, a criterion which specifies the minimum number of data points required for a region to be considered "dense". Observations that have a minimum number of other observations (specified by $MinPts$) within a given Eps of itself are known as *core points*. This is referred to as the *core point criterion*. If an observation lies within the Eps 'radius' of a core point, but does not satisfy the $MinPts$ criterion, it is referred to as a *border point*. On the other hand, observations that are neither core points nor connected to any core point are said to be outliers.

DBSCAN aims to construct dense regions (clusters) by making use of core points and their neighbourhoods. However, prior to explaining the DBSCAN clustering procedure, it is necessary to define the concepts of *directly-density reachable* and *density-connectedness*.

Definition 13 Directly-Density Reachable

A point $\boldsymbol{x}_i$ is said to be directly-density reachable from another point $\boldsymbol{x}_r$, $i, r = 1, ..., N$, $i \neq r$ with respect to the parameters Eps and MinPts if:

- *$\boldsymbol{x}_r$ is a core point;*
- *$\boldsymbol{x}_i$ lies within the Eps-neighbourhood of $\boldsymbol{x}_r$ i.e. $d(\boldsymbol{x}_i, \boldsymbol{x}_r) \leq Eps$, where d is a distance metric.*

Definition 14 Density-Connectedness

Two points $\boldsymbol{x}_i$ *and* $\boldsymbol{x}_r$, $i, r = 1, ..., N$, $i \neq r$, *are density-connected if* $\exists$ *a sequence of points* $(\boldsymbol{x}_{i_1}, ..., \boldsymbol{x}_{i_v})$ *in* $\mathbb{X}$, *such that:*

- $\boldsymbol{x}_{i_1} = \boldsymbol{x}_i$,
- $\boldsymbol{x}_{i_v} = \boldsymbol{x}_r$,
- *each point in the sequence is directly-density reachable from the previous one.*

The closure of density-connectedness is then used to find connected dense regions as clusters, where every closed set is a density-based cluster. Let D be the set of all observation from $\mathbb{X}$, i.e. $D = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$, then a subset $C \subseteq D$ is a cluster if:

1. For any two points $\boldsymbol{x}_q, \boldsymbol{x}_r \in C$, $q, r = 1, ..., N$, $q \neq r$, $\boldsymbol{x}_q$ and $\boldsymbol{x}_r$ are density-connected.
2. $\nexists$ a point $\boldsymbol{x}_i \in C$ and another point $\boldsymbol{x}_{i'} \in (D - C)$ such that $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i'}$ are density-connected, $i, i' = 1, ..., N$, $i \neq i'$.

DBSCAN then finds the clusters in the following way. Initially all elements in the set D are labelled as "unvisited". An unvisited point $\boldsymbol{x}_i \in D$, is randomly selected and marked as "visited". The *Eps*-neighbourhood of $\boldsymbol{x}_i$, denoted $\mathcal{N}_{Eps}(\boldsymbol{x}_i)$, is examined. If the number of elements in the neighbourhood is at least $MinPts$, then $\boldsymbol{x}_i$ is identified as a core point, and a new cluster C is created with $\boldsymbol{x}_i$ as its initial member. All elements in $\mathcal{N}_{Eps}(\boldsymbol{x}_i)$ are added to a candidate set M, which will be used to expand the cluster.

The cluster expansion process is then carried out by first checking which observations in M are core points. If a point $\boldsymbol{x}_{i'} \in M$, $i \neq i'$, is also a core point, then its own *Eps*-neighbourhood is added to M, allowing the cluster to grow further. Observations that are within the *Eps*-neighbourhood of core points but do not satisfy the core point criterion are still included in the cluster but do not contribute to its expansion. This process is continued iteratively until no more observations can be added to C, at which point the cluster is complete. Once a cluster has been formed, the algorithm searches for the next cluster by randomly selecting another unvisited point from D and repeating the process. If the selected observation does not satisfy the core point criterion, it remains unclustered and is eventually classified as an outlier. This procedure is continued until all observations have been visited.

The DBSCAN parameters, *Eps* and $MinPts$, that lead to the identification of adequate clusters, needs to be specified by the user. Such parameters specifications are typically empirically determined. Nevertheless, according to the literature, when

$n_x = 2$, $MinPts$ is conventionally set to 5 (Ester et al., 1996), whereas when $n_x \geq 3$, $MinPts$ is set to $2n_x$ (Sander et al., 1998). To determine optimal value for Eps, the distance to the κ^{th}-nearest neighbour are plotted for each point $\boldsymbol{x}_i$ in the dataset, where $\kappa = MinPts - 1$. This approach involves sorting these distances in ascending order and plotting them against their indices. The "elbow" point in the resulting curve, i.e. where the slope significantly changes, indicates a suitable value for Eps. It is worth noting that different hyperparameter values might result in the discovery of different clusters.

K.3 HDBSCAN

DBSCAN can only provide a "flat" (i.e. non-hierarchical) labelling of observations, based on a global density threshold. Utilizing a single density threshold might not properly characterize data sets with clusters of varying densities and/or nester clusters. Consequently HDBSCAN was developed to cater for this issue by generating a complete density-based clustering hierarchy from which a simplified hierarchy made up of only the most significant clusters that can be easily extracted. Prior to discussing HDBSCAN, we proceed to give a brief overview of hierarchical clustering.

K.3.1 Hierarchical Clustering Methods

Hierarchical clustering methods aim to build a hierarchy of clusters typically represented as a tree-like diagram called a *dendrogram*. When dealing with these types of methods, there are two approaches that could be taken, namely *agglomerative* or *divisive*, based on how the hierarchy of clusters is formed.

In the case of the agglomerative approach, each element is initially regarded as a separate group. The elements or groups 'close' to each other are then combined until all the clusters have been merged into one, or a termination condition is satisfied. This is referred to as a bottom-up approach. Conversely, the divisive approach starts with all elements being in the same cluster and recursively splits clusters until each element is in a cluster by itself, or until the node is terminated. This is a top-down approach.

Cluster analysis based on hierarchical methods can be distance-based and/or density and continuous based. Apart from this, they have been extended to cater for clustering in subspaces as well. One disadvantage of hierarchical methods is that once a merge or split is implemented, it can never be undone. Having said that, this makes the method less computationally expensive as the algorithm is not concerned with the

combinatorial number of different choices. It is worth noting that methods that can be used to improve upon hierarchical method's quality have been developed, however, they do not correct any erroneous decisions (Han et al., 2012).

K.3.2 DBSCAN*

Before outlining the HDBSCAN method, it is necessary to introduce first, a formal definition of *Eps*-reachability is required:

Definition 15 ***Eps*-Reachable**
Two core points $\boldsymbol{x}_r$ and $\boldsymbol{x}_q$ are said to be Eps-reachable with respect to Eps and $MinPts$ if $\boldsymbol{x}_r \in \mathcal{N}(\boldsymbol{x}_q)$ and $\boldsymbol{x}_q \in \mathcal{N}(\boldsymbol{x}_r)$.

Now based on the definition of *Eps-reachability*, *core points*, *density connectedness*, and *cluster* (see Section K.2), an algorithm DBSCAN* can be devised, whereby clusters are identifies as components of a graph in which the elements in D are vertices and each pair of vertices is adjacent if and only if the corresponding data points are *Eps*-reachable with respect to user defined parameters *Eps* and $MinPts$. Non-core points are labelled as outliers.

K.3.3 HDBSCAN

HDBSCAN has a single input parameter $MinPts$. Different density levels in the generated density-based cluster hierarchy will then correspond to different values of the radius *Eps*. In order to properly formulate the density based hierarchy with respect to a value of $MinPts$, the concepts of *core distance*, *Eps-core point*, *mutual reachability distance* and *mutual reachability graph* will be defined.

Definition 16 Core Distance
The core distance of a data point $\boldsymbol{x}_r \in D$ with respect to $MinPts$, $d_{core}(\boldsymbol{x}_r)$, is the distance from $\boldsymbol{x}_r$ to its $MinPts$-nearest neighbour including $\boldsymbol{x}_r$.

Definition 17 ***Eps*-Core Point**
A data point $\boldsymbol{x}_r \in D$ is called an Eps-core point if $\forall Eps \geq d_{core}(\boldsymbol{x}_r)$, w.r.t $MinPts$.

Definition 18 Mutual Reachability Distance
The mutual reachability distance between two observations $\boldsymbol{x}_r$ and $\boldsymbol{x}_q$ in D with respect to $MinPts$ is defined as $d_{mReach}(\boldsymbol{x}_r, \boldsymbol{x}_q) = \max\{d_{core}(\boldsymbol{x}_r, d_{core}(\boldsymbol{x}_q), d(\boldsymbol{x}_r, \boldsymbol{x}_q))\}$.

Definition 19 Mutual Reachability Graph

A mutual reachability graph, $\mathcal{G}_{MinPts}$, is a complete graph in which the observations of D are vertices and the weight of each edge is the mutual reachability distance, w.r.t $MinPts$, between the respective pair of observations.

Suppose $\mathcal{G}_{MinPts,Eps} \subseteq \mathcal{G}_{MinPts}$ is a graph obtained by removing all edges from $\mathcal{G}_{MinPts}$ that have weights greater than Eps. From the definitions of a *cluster*, *Eps-core point* and *mutual reachability graph*, it is intuitive to infer that clusters according to DBSCAN* with respect to $MinPts$ and Eps are the connected components of Eps-core points in $\mathcal{G}_{MinPts,Eps}$. The remaining observations are outliers. As a result, all DBSCAN* partitions for $Eps \in [0, \infty)$, can be outputted in a nested, hierarchical way by discarding edges in decreasing order of weight from $\mathcal{G}_{MinPts}$. Thus, a hierarchical version of DBSCAN* could be implemented by an algorithm that starts by computing a Single-Linkage hierarchy to find connected components and outliers at each level. Campello et al. (2013) proposed a more efficient equivalent solution.

A density based cluster hierarchy has to represent the fact that an element $\boldsymbol{x}_o \in D$ is an outlier below the level l that corresponds to $\boldsymbol{x}_o$'s core distance. To demonstrate this in a dendrogram, Campello et al. (2013) proposed to include an additional dendrogram node for $\boldsymbol{x}_o$ at level l representing the cluster containing $\boldsymbol{x}_o$ at that level and higher. In order to directly build such a hierarchy, an extension of a minimal spanning tree of the mutual reachability graph $\mathcal{G}_{MinPts}$ was proposed, from which an extended dendrogram can be constructed by removing edges in decreasing order of weights. Specifically, the minimal spanning tree is extended by connecting each vertex $\boldsymbol{x}_o$ to itself (self loops) where the edge weight is set to $d_{core}(\boldsymbol{x}_o)$. Said "self edges" will then be considered when removing edges. The pseudocode for HDSBCAN, which has inputs $MinPts$ and a set of elements D, can be found in Algorithm 2.

Algorithm 2 HDBSCAN main steps (Campello et al., 2013)

1. Compute the core distance w.r.t. $MinPts$ for all data objects in $\mathcal{D}$.
2. Compute an MST of G_{MinPts}, the Mutual Reachability Graph.
3. Extend the MST to obtain MST_{ext}, by adding for each vertex a "self edge" with the core distance of the corresponding object as weight.
4. Extract the HDBSCAN hierarchy as a dendrogram from MST_{ext}:
   4.1. For the root of the tree assign all objects the same label (single "cluster").
   4.2. Iteratively remove all edges from MST_{ext} in decreasing order of weights (in case of ties, edges must be removed simultaneously):
      4.2.1. Before each removal, set the dendrogram scale value of the current hierarchical level as the weight of the edge(s) to be removed.
      4.2.2. After each removal, assign labels to the connected component(s) that contain(s) the end vertex(-ices) of the removed edge(s), to obtain the next hierarchical level: assign a new cluster label to a component if it still has at least one edge, else assign it a null label ("noise").

HDSBCAN generates a clustering tree that contains all DBSCAN* partitions, with respect to $MinPts$, in a hierarchical, nested way. Additionally, it contains nodes that indicate when an isolated data point changes from core to outlier. The result is referred to as the "HDBSCAN hierarchy". HBDSCAN runs in $\mathcal{O}(n_x \cdot N^2)$ time and requires either $\mathcal{O}(n_x \cdot N)$ or $\mathcal{O}(N^2)$ space, depending on whether distances are calculated on the fly or stored in a matrix (Campello et al., 2013).

K.3.4 Hierarchy Simplification

The HDBSCAN hierarchy can easily be visualised as a conventional dendrogram. Nevertheless, interpreting these plots for relatively large data sets is not straightforward, making the extraction of only "significant" clusters from a dendrogram a fundamental problem. Consequently, Campello et al. (2013) proposed a simplification of the HDBSCAN hierarchy based on a fundamental observation about estimates of the level sets of continuous-valued probability density functions. For a given probability density function, there exists any of the following possibilities for the evolution of the connected components of a continuous density level set when increasing the density level (decreasing *Eps* in this case). The component shrinks but stays connected, up to a density threshold at which either:

- the component is split up into smaller ones;
- the component disappears.

This observation can be applied to the HDBSCAN hierarchy by selecting only those hierarchical levels where new clusters emerge through a genuine split or where existing clusters vanish, as these represent the points of most significant structural

change in the clustering process. When decreasing *Eps*, the mere removal of outliers from a cluster does not constitute a true split; in such cases, the cluster simply shrinks and should therefore retain its original label.

This idea can be generalized by setting the minimum cluster size, m_{clSize}. When $m_{clSize} \leq 1$, components with less than m_{clSize} data points are disregarded and their disconnection from a cluster does not establish a "true" split. HDBSCAN can be adapted accordingly by altering Step 4.2.2 of Algorithm 2 as shown in Algorithm 3, whereby a connected component is considered *spurious* if it has less than m_{clSize} data points or for $m_{clSize} = 1$, if it is an isolated, non-dense data points (i.e. no edges). Any spurious component is labelled as an outlier and its removal from a larger component is not deemed as a cluster split. In practice, this can diminish the size of the hierarchy significantly. To make HDBSCAN similar to prior density-based methods, as well as to simplify its use, m_{clSize} can be set to $MinPts$. This turns $MinPts$ into a single parameter that serves as both a smoothing factor and a threshold for the cluster size. It is worth noting that HDBSCAN can extract an optimal non-hierarchical clustering from the hierarchy by maximizing cluster stability, labelling data points that do not belong to stable clusters as outliers.

Algorithm 3 HDBSCAN step 4.2.2 with (optional) parameter $m_{clSize} \geq 1$ (Campello et al., 2013)

4.2.2 After each removal (to obtain the next hierarchical level), process one at a time each cluster that contained the edge(s) just removed, by relabeling its resulting connected subcomponent(s):

Label *spurious* subcomponents as noise by assigning them the null label. If all subcomponents of a cluster are *spurious*, then the **cluster has disappeared**.

Else, if a single subcomponent of a cluster is *not spurious*, keep its original cluster label (**cluster has just shrunk**).

Else, if two or more subcomponents of a cluster are *not spurious*, assign new cluster labels to each of them (**"true" cluster split**).

K.4 Assessing Clustering Tendency and the Hopkins Statistic

Assessing clustering tendency aims to determine whether there exists a non-random structure within a given dataset that may lead to meaningful clusters. For instance, suppose a dataset is a set of uniformly distributed points, where the term *uniformly distributed* is not used in the statistical sense but describes a scenario where observation in said dataset are spread evenly and randomly across the feature space without any areas of high/low density. Although the clustering algorithm may return clusters, they are not considered meaningful because there does not exist a non-random structure within the data. It is worth noting that throughout this section, the term "uniformly distributed" will take the just stated definition.

One method to assessing the clustering tendency of a dataset is to measure the probability that the elements within said dataset are uniformly distributed. This can be done through the use of statistical tests for spatial randomness, such as the Hopkins statistic (Hopkins & Skellam, 1954). Let D be a set of all elements from $\mathbb{X}$, i.e. $D = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ and let $\tilde{D} = \{\tilde{\boldsymbol{x}}_1, ..., \tilde{\boldsymbol{x}}_{\tilde{N}}\}$ be the set of all elements randomly sampled without replacement from D, $\tilde{N} < N$. Here $\tilde{N}$ is a small fraction of N, for example 10% (Lawson & Jurs, 1990). A set $\tilde{U} = \{\tilde{\boldsymbol{u}}_1, ..., \tilde{\boldsymbol{u}}_{\tilde{N}}\}$ of uniformly randomly distributed elements is generated. The objective is to determine how different $\tilde{D}$ is from $\tilde{U}$. This can be achieved through computing the Hopkins statistic, H:

$$H = \frac{\sum_{\tilde{i}} \tilde{u}_{\tilde{i}}^{n_x}}{\sum_{\tilde{i}} (\tilde{u}_{\tilde{i}}^{n_x} + \tilde{w}_{\tilde{i}}^{n_x})}, \tag{K.5}$$

where n_x represents the number of features, $\tilde{u}_{\tilde{i}}$ is the minimum distance from $\tilde{\boldsymbol{u}}_{\tilde{i}} \in \tilde{U}$, $\tilde{i} = 1, ..., \tilde{N}$ to its nearest neighbour in D, and $\tilde{w}_{\tilde{i}}$ is the minimum distance of $\tilde{\boldsymbol{x}}_{\tilde{i}} \in \tilde{D} \subseteq D$, to its nearest neighbour $\boldsymbol{x}_i \in D$, $\tilde{\boldsymbol{x}}_{\tilde{i}} \neq \boldsymbol{x}_i$. The distances $\tilde{u}_{\tilde{i}}$ and $\tilde{w}_{\tilde{i}}$ are calculated using a distance metric like the Euclidean distance.

It is convention to compute H multiple times and take the average due to sampling variability. Additionally, it is worth noting that under the spatial randomness hypothesis, i.e. the data is generated by Poisson point process and thus, uniformly randomly distributed, H has a $Beta(\tilde{N}, \tilde{N})$ distribution and thus, will always fall within the range [0,1]. The Hopkins statistic is then interpreted in the following way:

- Values of $H < 0.5$ indicate that the elements in D are regularly spaced, suggesting repulsion between elements.
- Values of H close to 0.5 indicate spatial randomness of the elements in D.
- When $0.5 < H < 0.75$, there is a potential clustering tendency among the elements in D.
- Values of $H > 0.75$ suggest a strong clustering tendency at the 90% confidence level (Lawson & Jurs, 1990).

K.5 Clustering Evaluation

After a clustering technique has been applied to the data, it is important to assess the quality of the outputted clustering. This can be accomplished through various methods depending on the task at hand. When clustering is used in simulation studies, *extrinsic methods* are typically employed. These methods rely on ground truth labels to

evaluate the clustering quality. Nevertheless, in the application presented in Chapter 5, clustering algorithms are applied to real-world data where cluster labels are not available and so, *intrinsic methods* are used instead. Intrinsic methods assess the clustering quality by examining the compactness and separation of clusters based on some objective criteria, without any knowledge on how the true partition should look like.

There are numerous intrinsic metrics that can be used, depending on the nature of the data and clusters, as well as the algorithms used. One of the most commonly used intrinsic validity index was proposed by Dunn in 1973. Since then, there have been several extensions and generalizations of Dunn's index that employ different measures of compactness and separation between clusters. One notable generalization was developed by Pal & Biswas (1997) in order to facilitate the evaluation of clusterings when the data contains irregular or complex shaped clusters. This was accomplished by leveraging graph theory concepts, especially through the used of the Gabriel graph which will be explained shortly. Ilc (2012) then modified this generalized Dunn's index to improve its ability to correctly identify such clusters. Having introduced the concept of clustering evaluation, we will now proceed to outline the relevant theory relating to Gabriel graphs, the generalized Dunn index and its modification.

K.5.1 Gabriel Graph

Let $D = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ be a set of n_x-dimensional data points, $\boldsymbol{x}_i \in \mathbb{R}^{n_x}$, $i = 1, ..., N$, and let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote a graph. For every element $\boldsymbol{x}_i$ there is a vertex $v_i \in \mathcal{V}$, that is an abstraction in the graph $\mathcal{G}$, thus $|D| = |\mathcal{V}| = |N|$. The proximity of two vertices $v_i, v_{i'} \in \mathcal{V}$ is defined as the Euclidean distance between their associated pair of elements, i.e. let $d_{EUC}(\boldsymbol{x}_i, \boldsymbol{x}_{i'})$ be linked by the edge $e_q = \{v_i, v_{i'}\}$ and let $\mathcal{G}$ be an undirected graph weighted by the Euclidean distance between the data points. Therefore, the weight of the edge $e_q = \{v_i, v_{i'}\}$ is calculated as $w(e_q) = d_{EUC}(\boldsymbol{x}_i, \boldsymbol{x}_{i'})$.

The Gabriel graph is a graph in which there exists an edge $e_q = \{v_i, v_{i'}\}$ such that:

$$d^2_{EUC}(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) < d^2_{EUC}(\boldsymbol{x}_i, \boldsymbol{x}_r) + d^2_{EUC}(\boldsymbol{x}_r, \boldsymbol{x}_{i'}), \tag{K.6}$$

$\forall r : v_r \in \mathcal{V}$, $r \neq i$, $r \neq i'$. In other words, vertices v_i and $v_{i'}$ are connected if there does not exist any other vertex v_r, such that its associated observation $\boldsymbol{x}_r$ would lie in the n_x-dimensional hyperplane with diameter $d_{EUC}(\boldsymbol{x}_i, \boldsymbol{x}_{i'})$ and its centre in $\boldsymbol{x}_i + \frac{(\boldsymbol{x}_{i'} - \boldsymbol{x}_i)}{2}$. The notion of a Gabriel graph is illustrated in Figure K.1.

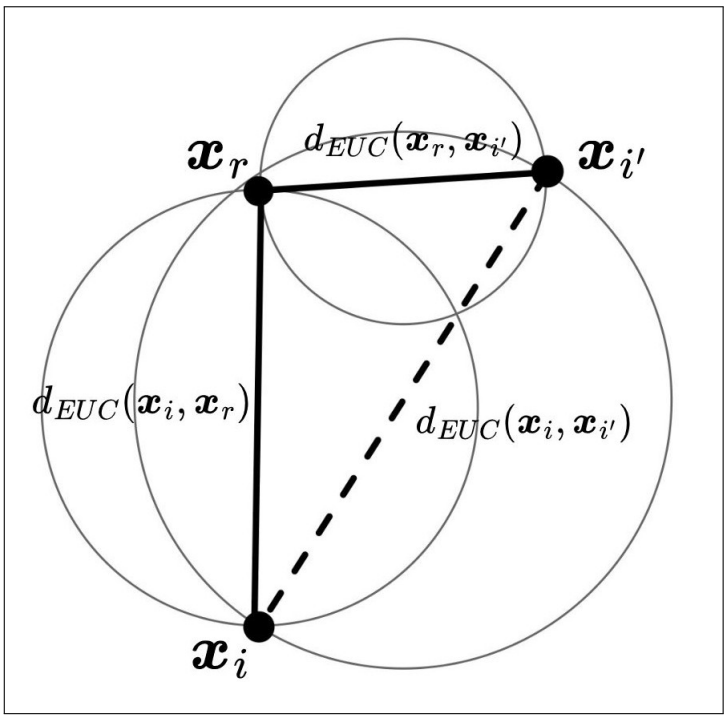

Figure K.1: A construction of the Gabriel graph on three data points. Solid line represents an edge between two vertices (adapted from Ilc (2012)).

K.5.2 Dunn's Index and its Generalization

Assume a set of elements D has been divided into K clusters, C_k, $k = 1, ..., K$ and obtained the following partition $\mathcal{C}_K = \{C_1, ..., C_K\}$ such that $D = \bigcup_{k=1}^{K} C_k$ and $C_k \bigcap C_{k'} = \emptyset$, $k \neq k'$, $C_k \neq \emptyset$. The Dunn's validity index of the partition $\mathcal{C}_K$ can be calculated as:

$$DN(\mathcal{C}_K) = \frac{\min_{1 \leq k \leq K}\{\min_{1 \leq k' \leq K}\{dist(C_k, C_{k'})\}\}}{\max_{1 \leq l \leq K}\{diam(C_l)\}}, \tag{K.7}$$

where

$$diam(C_k) = \max_{\boldsymbol{x}_i, \boldsymbol{x}_{i'} \in C_k} \{d_{EUC}(\boldsymbol{x}_i, \boldsymbol{x}_{i'})\} \tag{K.8}$$

and

$$dist(C_k, C_{k'}) = \min_{\boldsymbol{x}_i \in C_k, \boldsymbol{x}_i \in C_{k'}, i \neq i} \{d_{EUC}(\boldsymbol{x}_i, \boldsymbol{x}_{i'})\}. \tag{K.9}$$

The higher the value of $DN(\mathcal{C}_K)$, the more compact and well-separated clusters in the partition $\mathcal{C}_K$ are. Thus, the partitions for different number of clusters K, or for any other parameter of the clustering algorithm may be computed, and the partition that maximizes Dunn's index is considered to be the optimal solution. It is worth noting that the complexity of Dunn's index is $\mathcal{O}(n_x \cdot N^2 + K^2)$.

Bezdek & Pal (1998) argue that the definitions of the inter cluster diameter

(Equation (K.13)) and the between cluster distance (Equation (K.14)) are too sensitive to data that contains outliers and are also unreliable for the validation of non-spherical clusters (Ilc, 2012). In order to combat these issues, Bezdek & Pal (1998) enhanced the index using the Gabriel graph which will be referred to as the generalized Dunn's index. The generalization works by representing the elements $\boldsymbol{x}_i \in D$ with vertices $v_i \in \mathcal{V}$ in the Gabriel graph and to redefine the cluster diameter and between cluster distance as:

$$diam_G(C_k) = \max_{1 \leq q \leq |\mathcal{E}_i|} \{e_q\}, \quad e_q \in \mathcal{E}_i, \tag{K.10}$$

and

$$dist_G(C_k, C_{k'}) = d_{EUC}(mean(C_k), mean(C_{k'})), \tag{K.11}$$

where $\mathcal{E}_i$ denotes the set of edges in the Gabriel graph, such that every edge $e_q \in \mathcal{E}_i$ links a pair of vertices both belonging to the cluster C_k and $mean(C_k)$ denotes the mean value of all the observations in the cluster C_k. The generalized Dunn's index of the clustering $\mathcal{C}_K$, $DN_G(\mathcal{C}_K)$ is calculated as in Equation (K.7) replacing *diam* with $diam_G$ and *dist* with $dist_G$. Analogous to before, the higher value of $DN_G(\mathcal{C}_K)$ indicates the better partition. It is worth noting that the computational complexity of the generalized Dunn's index is $\mathcal{O}(n_x \cdot N^3 + n_x \cdot K^2 + N)$.

K.5.3 The Modified Dunn's Index

The modified Dunn's index improves upon the conventional generalized Dunn's index by modifying the calculation of the cluster diameter and distance between clusters. The principal idea is to define the distance between a pair of observations in terms of the shortest path between them in the Gabriel graph. Let $\mathcal{G}(\mathcal{V}, \mathcal{E})$ be the Gabriel graph built on D, a *path* p between v_i and v_i' is a sequence of vertices, such that there exists an edge between each of the two successive vertices. A path that has no repeated vertices is referred to as a *simple path* and the length l_p is equal to the number of edges on p. Denote the edge on a path p as e_t^p, $t = 1, ..., l_p$ and suppose there are P possible paths between a pair of vertices $v_i, v_{i'} \in \mathcal{V}$. The distance between the elements $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i'}$ is defined as:

$$d_S(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) = \min_{1 \leq p \leq P} \Big\{ \sum_{t=1}^{l_p} w(e_t^p) \Big\}. \tag{K.12}$$

Here $w(e_t^p)$ represents the weight of the edge on the path p between v_i and $v_{i'}$ and it is equal to $d_{EUC}(\boldsymbol{x}_i, \boldsymbol{x}_{i'})$. Figure K.2 shows the difference between the distances, d_{EUC} and d_S, between the pair of observations $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i'}$, where solid lines represent existing edges

in the Gabriel graph. In this simple case, $d_S(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) = d_{EUC}(\boldsymbol{x}_i, \boldsymbol{x}_r) + d_{EUC}(\boldsymbol{x}_r, \boldsymbol{x}_{i'})$. To determine the shortest paths between all pairs of vertices in $\mathcal{V}$, Johnson's algorithm (Johnson, 1977) is employed. The definitions of the cluster diameter and the between cluster distance are now:

$$diam_S(C_k) = \max_{\boldsymbol{x}_i, \boldsymbol{x}_{i'} \in C_k} \{d_S(\boldsymbol{x}_i, \boldsymbol{x}_{i'})\} \tag{K.13}$$

and

$$dist_S(C_k, C_{k'}) = \min_{\boldsymbol{x}_i \in C_k, \boldsymbol{x}_i \in C_{k'}, i \neq i} \{d_S(\boldsymbol{x}_i, \boldsymbol{x}_{i'})\}. \tag{K.14}$$

Here d is a non-negative real number for all the pairs of observations in D, since in the Gabriel graph, there always exists a path between any two vertices (Ilc, 2012). The modified Dunn's index of a clustering $\mathcal{C}_k$, $DN_S(\mathcal{C}_K)$ is as in Equation (K.7) but replacing $diam$ with $diam_S$ and $dist$ with $dist_S$. Once again, the larger the value of $DN_S(\mathcal{C}_K)$, the better the partition $\mathcal{C}_K$ is considered to be. It is worthing noting that the complexity of the modified Dunn's index is $\mathcal{O}(n_x \cdot N^3 + N^2 \log N + N^2 + K^2)$.

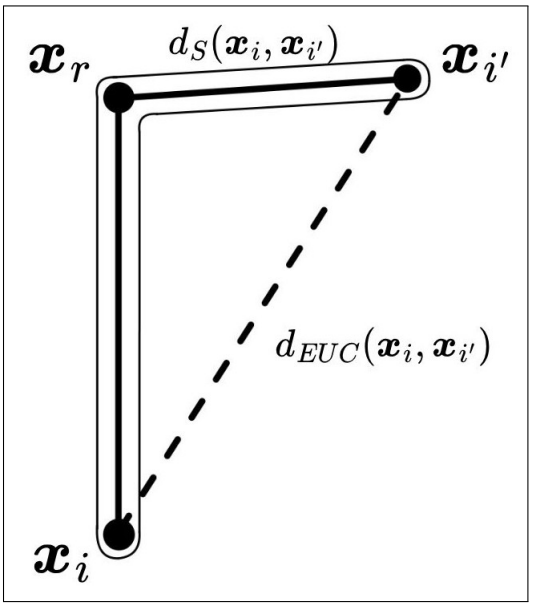

Figure K.2: The Euclidean distance d_{EUC} (dashed line) and the shortest distance d_S in the Gabriel graph (solid line highlighted with a contour) between the points $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i'}$(adapted from Ilc (2012)).

Appendix L

Summary of Performance Metrics for Hyperparameter Tuning using Edgeless Graph Model on Reduced Dataset

Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
16	0.00001	0.2	0.42	0.26	1.10
16	0.00001	0.3	0.57	0.50	1.53
16	0.00001	0.4	0.78	0.82	1.96
16	0.00001	0.5	0.80	0.84	1.98
16	0.0001	0.2	0.39	0.22	1.02
16	0.0001	0.3	0.38	0.22	1.02
16	0.0001	0.4	0.37	0.21	0.99
16	0.0001	0.5	0.73	0.73	1.85
16	0.001	0.2	0.42	0.26	1.10
16	0.001	0.3	0.38	0.23	1.04
16	0.001	0.4	0.39	0.23	1.04
16	0.001	0.5	0.37	0.22	1.02
16	0.01	0.2	0.37	0.23	1.04
16	0.01	0.3	0.37	0.23	1.04
16	0.01	0.4	0.37	0.22	1.02
16	0.01	0.5	0.37	0.23	1.04
32	0.00001	0.2	0.46	0.33	1.24
32	0.00001	0.3	0.52	0.44	1.44
32	0.00001	0.4	0.66	0.65	1.75
32	0.00001	0.5	0.41	0.25	1.08
32	0.0001	0.2	0.38	0.22	1.02
32	0.0001	0.3	0.37	0.21	0.99
32	0.0001	0.4	0.46	0.32	1.22
32	0.0001	0.5	0.85	0.94	2.10
32	0.001	0.2	0.40	0.24	1.06
32	0.001	0.3	0.38	0.23	1.04
32	0.001	0.4	0.37	0.22	1.02
32	0.001	0.5	0.37	0.23	1.04
32	0.01	0.2	0.38	0.22	1.02
32	0.01	0.3	0.37	0.22	1.02
32	0.01	0.4	0.37	0.23	1.04
32	0.01	0.5	0.37	0.23	1.04

Table L.1: Validation performance metrics for latitude using the Edgeless Graph Architecture on reduced dataset (part 1).

Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
64	0.00001	0.2	0.51	0.41	1.39
64	0.00001	0.3	0.59	0.52	1.56
64	0.00001	0.4	0.65	0.60	1.68
64	0.00001	0.5	0.80	0.85	2.00
64	0.0001	0.2	0.39	0.22	1.02
64	0.0001	0.3	0.40	0.23	1.04
64	0.0001	0.4	0.43	0.28	1.15
64	0.0001	0.5	0.84	0.93	2.09
64	0.001	0.2	0.38	0.22	1.02
64	0.001	0.3	0.37	0.23	1.04
64	0.001	0.4	0.37	0.23	1.04
64	0.001	0.5	0.37	0.23	1.04
64	0.01	0.2	0.39	0.23	1.04
64	0.01	0.3	0.37	0.22	1.02
64	0.01	0.4	0.39	0.24	1.06
64	0.01	0.5	0.38	0.23	1.04

Table L.2: Validation performance metrics for latitude using the Edgeless Graph Architecture on reduced dataset (part 2).

Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
16	0.00001	0.2	0.50	0.50	1.07
16	0.00001	0.3	0.76	0.89	1.43
16	0.00001	0.4	1.04	1.45	1.82
16	0.00001	0.5	0.79	0.94	1.47
16	0.0001	0.2	0.46	0.44	1.00
16	0.0001	0.3	0.47	0.44	1.00
16	0.0001	0.4	0.49	0.46	1.03
16	0.0001	0.5	1.07	1.46	1.83
16	0.001	0.2	0.48	0.46	1.03
16	0.001	0.3	0.46	0.44	1.00
16	0.001	0.4	0.46	0.44	1.00
16	0.001	0.5	0.46	0.44	1.00
16	0.01	0.2	0.46	0.44	1.00
16	0.01	0.3	0.46	0.44	1.00
16	0.01	0.4	0.46	0.44	1.00
16	0.01	0.5	0.46	0.44	1.00
32	0.00001	0.2	0.54	0.53	1.10
32	0.00001	0.3	0.73	0.81	1.36
32	0.00001	0.4	1.07	1.51	1.86
32	0.00001	0.5	0.88	1.11	1.59
32	0.0001	0.2	0.48	0.46	1.03
32	0.0001	0.3	0.47	0.45	1.01
32	0.0001	0.4	0.54	0.53	1.10
32	0.0001	0.5	1.23	1.89	2.08
32	0.001	0.2	0.46	0.44	1.00
32	0.001	0.3	0.46	0.44	1.00
32	0.001	0.4	0.46	0.44	1.00
32	0.001	0.5	0.46	0.44	1.00
32	0.01	0.2	0.47	0.44	1.00
32	0.01	0.3	0.46	0.44	1.00
32	0.01	0.4	0.46	0.44	1.00
32	0.01	0.5	0.46	0.44	1.00

Table L.3: Validation performance metrics for longitude using the Edgeless Graph Architecture on reduced dataset (part 1).

Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
64	0.00001	0.2	0.64	0.67	1.24
64	0.00001	0.3	0.85	1.02	1.53
64	0.00001	0.4	0.72	0.81	1.36
64	0.00001	0.5	0.55	0.55	1.12
64	0.0001	0.2	0.48	0.46	1.03
64	0.0001	0.3	0.49	0.47	1.04
64	0.0001	0.4	0.53	0.53	1.10
64	0.0001	0.5	1.21	1.88	2.07
64	0.001	0.2	0.46	0.44	1.00
64	0.001	0.3	0.46	0.44	1.00
64	0.001	0.4	0.46	0.44	1.00
64	0.001	0.5	0.46	0.44	1.00
64	0.01	0.2	0.46	0.44	1.00
64	0.01	0.3	0.46	0.44	1.00
64	0.01	0.4	0.47	0.44	1.00
64	0.01	0.5	0.47	0.45	1.01

Table L.4: Validation performance metrics for longitude using the Edgeless Graph Architecture on reduced dataset (part 2).

Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
16	0.00001	0.2	7.15	63.67	0.99
16	0.00001	0.3	7.16	64.71	0.99
16	0.00001	0.4	7.14	64.57	0.99
16	0.00001	0.5	6.97	63.79	0.99
16	0.0001	0.2	6.96	61.04	0.96
16	0.0001	0.3	7.08	62.67	0.98
16	0.0001	0.4	7.11	63.41	0.98
16	0.0001	0.5	7.23	63.40	0.98
16	0.001	0.2	6.62	64.30	0.99
16	0.001	0.3	6.31	54.70	0.91
16	0.001	0.4	6.50	56.60	0.93
16	0.001	0.5	6.72	58.67	0.95
16	0.01	0.2	7.35	68.19	1.02
16	0.01	0.3	7.37	68.70	1.02
16	0.01	0.4	7.35	68.24	1.02
16	0.01	0.5	7.36	68.46	1.02
32	0.00001	0.2	7.24	66.14	1.00
32	0.00001	0.3	7.52	69.77	1.03
32	0.00001	0.4	7.03	62.16	0.97
32	0.00001	0.5	7.39	73.04	1.06
32	0.0001	0.2	6.91	59.27	0.95
32	0.0001	0.3	7.10	63.55	0.98
32	0.0001	0.4	7.17	62.75	0.98
32	0.0001	0.5	7.02	62.37	0.97
32	0.001	0.2	6.20	52.95	0.90
32	0.001	0.3	6.45	55.90	0.92
32	0.001	0.4	6.90	59.75	0.95
32	0.001	0.5	7.05	62.75	0.98
32	0.01	0.2	6.90	59.42	0.95
32	0.01	0.3	7.00	61.98	0.97
32	0.01	0.4	7.34	68.00	1.02
32	0.01	0.5	7.37	68.74	1.02

Table L.5: Validation performance metrics for depth using the Edgeless Graph Architecture on reduced dataset (part 1).

Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
64	0.00001	0.2	7.67	74.10	1.06
64	0.00001	0.3	6.90	62.92	0.98
64	0.00001	0.4	7.33	70.61	1.04
64	0.00001	0.5	7.33	75.74	1.07
64	0.0001	0.2	6.98	61.38	0.97
64	0.0001	0.3	7.30	64.96	1.00
64	0.0001	0.4	7.08	61.81	0.97
64	0.0001	0.5	7.24	68.04	1.02
64	0.001	0.2	6.51	58.53	0.94
64	0.001	0.3	6.94	60.55	0.96
64	0.001	0.4	7.03	61.82	0.97
64	0.001	0.5	7.20	65.08	1.00
64	0.01	0.2	6.24	52.17	0.89
64	0.01	0.3	6.22	52.19	0.89
64	0.01	0.4	7.14	64.11	0.99
64	0.01	0.5	7.10	63.23	0.98

Table L.6: Validation performance metrics for depth using the Edgeless Graph Architecture on reduced dataset (part 2).

Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
16	0.00001	0.2	0.30	0.19	1.27
16	0.00001	0.3	0.34	0.23	1.40
16	0.00001	0.4	0.34	0.22	1.37
16	0.00001	0.5	0.33	0.22	1.37
16	0.0001	0.2	0.27	0.14	1.09
16	0.0001	0.3	0.27	0.14	1.09
16	0.0001	0.4	0.27	0.14	1.09
16	0.0001	0.5	0.33	0.22	1.37
16	0.001	0.2	0.28	0.14	1.09
16	0.001	0.3	0.27	0.13	1.05
16	0.001	0.4	0.27	0.13	1.05
16	0.001	0.5	0.27	0.14	1.09
16	0.01	0.2	0.27	0.14	1.09
16	0.01	0.3	0.27	0.14	1.09
16	0.01	0.4	0.27	0.13	1.05
16	0.01	0.5	0.27	0.14	1.09
32	0.00001	0.2	0.29	0.18	1.24
32	0.00001	0.3	0.32	0.21	1.34
32	0.00001	0.4	0.33	0.22	1.37
32	0.00001	0.5	0.32	0.21	1.34
32	0.0001	0.2	0.27	0.14	1.09
32	0.0001	0.3	0.27	0.14	1.09
32	0.0001	0.4	0.28	0.16	1.17
32	0.0001	0.5	0.34	0.23	1.40
32	0.001	0.2	0.28	0.13	1.05
32	0.001	0.3	0.27	0.13	1.05
32	0.001	0.4	0.27	0.13	1.05
32	0.001	0.5	0.27	0.14	1.09
32	0.01	0.2	0.27	0.14	1.09
32	0.01	0.3	0.27	0.14	1.09
32	0.01	0.4	0.27	0.14	1.09
32	0.01	0.5	0.27	0.14	1.09

Table L.7: Validation performance metrics for magnitude using the Edgeless Graph Architecture on reduced dataset (part 1).

Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
64	0.00001	0.2	0.33	0.22	1.37
64	0.00001	0.3	0.32	0.21	1.34
64	0.00001	0.4	0.30	0.19	1.27
64	0.00001	0.5	0.30	0.18	1.24
64	0.0001	0.2	0.27	0.13	1.05
64	0.0001	0.3	0.29	0.18	1.24
64	0.0001	0.4	0.33	0.22	1.37
64	0.0001	0.5	0.34	0.23	1.40
64	0.001	0.2	0.28	0.14	1.09
64	0.001	0.3	0.27	0.13	1.05
64	0.001	0.4	0.27	0.14	1.09
64	0.001	0.5	0.27	0.14	1.09
64	0.01	0.2	0.27	0.14	1.09
64	0.01	0.3	0.27	0.14	1.09
64	0.01	0.4	0.27	0.13	1.05
64	0.01	0.5	0.27	0.14	1.09

Table L.8: Validation performance metrics for magnitude using the Edgeless Graph Architecture on reduced dataset (part 2).

Batch Size	LR	Dropout Prob	NRMSE
16	0.00001	0.2	1.11
16	0.00001	0.3	1.34
16	0.00001	0.4	1.54
16	0.00001	0.5	1.45
16	0.0001	0.2	1.02
16	0.0001	0.3	1.02
16	0.0001	0.4	1.02
16	0.0001	0.5	1.51
16	0.001	0.2	1.05
16	0.001	0.3	1.00
16	0.001	0.4	1.01
16	0.001	0.5	1.01
16	0.01	0.2	1.04
16	0.01	0.3	1.04
16	0.01	0.4	1.02
16	0.01	0.5	1.04
32	0.00001	0.2	1.15
32	0.00001	0.3	1.29
32	0.00001	0.4	1.49
32	0.00001	0.5	1.27
32	0.0001	0.2	1.02
32	0.0001	0.3	1.02
32	0.0001	0.4	1.12
32	0.0001	0.5	1.64
32	0.001	0.2	1.00
32	0.001	0.3	1.00
32	0.001	0.4	1.01
32	0.001	0.5	1.03
32	0.01	0.2	1.02
32	0.01	0.3	1.02
32	0.01	0.4	1.04
32	0.01	0.5	1.04

Table L.9: Validation performance metrics for the four target variables using the Edgeless Graph Architecture on reduced dataset (part 1).

Batch Size	LR	Dropout Prob	NRMSE
64	0.00001	0.2	1.26
64	0.00001	0.3	1.35
64	0.00001	0.4	1.34
64	0.00001	0.5	1.36
64	0.0001	0.2	1.01
64	0.0001	0.3	1.08
64	0.0001	0.4	1.15
64	0.0001	0.5	1.64
64	0.001	0.2	1.01
64	0.001	0.3	1.01
64	0.001	0.4	1.03
64	0.001	0.5	1.03
64	0.01	0.2	1.01
64	0.01	0.3	1.00
64	0.01	0.4	1.03
64	0.01	0.5	1.03

Table L.10: Validation performance metrics for the four target variables using the Edgeless Graph Architecture on reduced dataset (part 2).

Appendix M

Summary of Performance Metrics for Hyperparameter Tuning using Dynamic Edges GNN Model on Reduced Dataset

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
3	16	0.00001	0.2	0.97	1.16	2.33
3	16	0.00001	0.3	0.84	0.93	2.09
3	16	0.00001	0.4	0.37	0.22	1.02
3	16	0.00001	0.5	0.64	0.60	1.68
3	16	0.0001	0.2	0.99	1.19	2.36
3	16	0.0001	0.3	0.98	1.18	2.35
3	16	0.0001	0.4	0.98	1.17	2.34
3	16	0.0001	0.5	0.97	1.15	2.32
3	16	0.001	0.2	0.42	0.30	1.19
3	16	0.001	0.3	0.50	0.36	1.30
3	16	0.001	0.4	0.37	0.23	1.04
3	16	0.001	0.5	0.38	0.23	1.04
3	16	0.01	0.2	0.37	0.23	1.04
3	16	0.01	0.3	0.37	0.23	1.04
3	16	0.01	0.4	0.37	0.23	1.04
3	16	0.01	0.5	0.37	0.22	1.02
3	32	0.00001	0.2	0.90	1.03	2.20
3	32	0.00001	0.3	0.85	0.93	2.09
3	32	0.00001	0.4	0.65	0.62	1.70
3	32	0.00001	0.5	0.52	0.41	1.39
3	32	0.0001	0.2	0.99	1.19	2.36
3	32	0.0001	0.3	0.99	1.19	2.36
3	32	0.0001	0.4	0.97	1.16	2.33
3	32	0.0001	0.5	0.94	1.10	2.27
3	32	0.001	0.2	0.36	0.21	0.99
3	32	0.001	0.3	0.37	0.22	1.02
3	32	0.001	0.4	0.37	0.23	1.04
3	32	0.001	0.5	0.38	0.25	1.08
3	32	0.01	0.2	0.37	0.22	1.02
3	32	0.01	0.3	0.37	0.23	1.04
3	32	0.01	0.4	0.37	0.23	1.04
3	32	0.01	0.5	0.37	0.22	1.02

Table M.1: Validation performance metrics for latitude using the Dynamic Edges GNN Architecture on reduced dataset (part 1).

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
3	64	0.00001	0.2	0.88	0.98	2.14
3	64	0.00001	0.3	0.64	0.60	1.68
3	64	0.00001	0.4	0.37	0.23	1.04
3	64	0.00001	0.5	0.49	0.36	1.30
3	64	0.0001	0.2	0.85	0.95	2.11
3	64	0.0001	0.3	0.98	1.18	2.35
3	64	0.0001	0.4	0.95	1.12	2.29
3	64	0.0001	0.5	0.87	0.96	2.12
3	64	0.001	0.2	0.42	0.30	1.19
3	64	0.001	0.3	0.37	0.22	1.02
3	64	0.001	0.4	0.39	0.24	1.06
3	64	0.001	0.5	0.99	1.19	2.36
3	64	0.01	0.2	0.41	0.27	1.12
3	64	0.01	0.3	0.39	0.24	1.06
3	64	0.01	0.4	0.39	0.25	1.08
3	64	0.01	0.5	0.38	0.23	1.04
4	16	0.00001	0.2	0.87	0.98	2.14
4	16	0.00001	0.3	0.91	1.04	2.21
4	16	0.00001	0.4	0.78	0.82	1.96
4	16	0.00001	0.5	0.36	0.21	0.99
4	16	0.0001	0.2	0.38	0.22	1.02
4	16	0.0001	0.3	0.99	1.19	2.36
4	16	0.0001	0.4	0.98	1.18	2.35
4	16	0.0001	0.5	0.97	1.16	2.33
4	16	0.001	0.2	0.36	0.22	1.02
4	16	0.001	0.3	0.36	0.22	1.02
4	16	0.001	0.4	0.36	0.22	1.02
4	16	0.001	0.5	0.37	0.22	1.02
4	16	0.01	0.2	0.37	0.23	1.04
4	16	0.01	0.3	0.37	0.23	1.04
4	16	0.01	0.4	0.37	0.23	1.04
4	16	0.01	0.5	0.37	0.23	1.04
4	32	0.00001	0.2	0.91	1.05	2.22
4	32	0.00001	0.3	0.91	1.03	2.20
4	32	0.00001	0.4	0.64	0.59	1.66
4	32	0.00001	0.5	0.44	0.30	1.19
4	32	0.0001	0.2	0.38	0.21	0.99
4	32	0.0001	0.3	0.98	1.18	2.35
4	32	0.0001	0.4	0.97	1.15	2.32
4	32	0.0001	0.5	0.92	1.07	2.24
4	32	0.001	0.2	1.83	5.44	5.05
4	32	0.001	0.3	0.37	0.22	1.02
4	32	0.001	0.4	0.37	0.22	1.02
4	32	0.001	0.5	0.99	1.19	2.36
4	32	0.01	0.2	0.38	0.22	1.02
4	32	0.01	0.3	0.38	0.23	1.04
4	32	0.01	0.4	0.37	0.23	1.04
4	32	0.01	0.5	0.37	0.23	1.04

Table M.2: Validation performance metrics for latitude using the Dynamic Edges GNN Architecture on reduced dataset (part 2).

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
4	64	0.00001	0.2	0.90	1.02	2.19
4	64	0.00001	0.3	0.90	1.02	2.19
4	64	0.00001	0.4	0.94	1.09	2.26
4	64	0.00001	0.5	0.40	0.25	1.08
4	64	0.0001	0.2	0.99	1.19	2.36
4	64	0.0001	0.3	0.98	1.17	2.34
4	64	0.0001	0.4	0.96	1.14	2.31
4	64	0.0001	0.5	0.93	1.08	2.25
4	64	0.001	0.2	0.37	0.21	0.99
4	64	0.001	0.3	0.96	1.13	2.30
4	64	0.001	0.4	0.99	1.19	2.36
4	64	0.001	0.5	0.99	1.19	2.36
4	64	0.01	0.2	0.40	0.24	1.06
4	64	0.01	0.3	0.41	0.27	1.12
4	64	0.01	0.4	0.37	0.22	1.02
4	64	0.01	0.5	0.38	0.22	1.02

Table M.3: Validation performance metrics for latitude using the Dynamic Edges GNN Architecture on reduced dataset (part 3).

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
3	16	0.00001	0.2	1.59	2.97	2.61
3	16	0.00001	0.3	1.56	2.87	2.56
3	16	0.00001	0.4	1.03	1.41	1.80
3	16	0.00001	0.5	0.98	1.31	1.73
3	16	0.0001	0.2	1.62	3.07	2.65
3	16	0.0001	0.3	1.63	3.10	2.66
3	16	0.0001	0.4	1.62	3.05	2.64
3	16	0.0001	0.5	1.59	2.95	2.60
3	16	0.001	0.2	0.58	0.57	1.14
3	16	0.001	0.3	0.57	0.65	1.22
3	16	0.001	0.4	0.45	0.43	0.99
3	16	0.001	0.5	0.47	0.44	1.00
3	16	0.01	0.2	0.46	0.44	1.00
3	16	0.01	0.3	0.46	0.44	1.00
3	16	0.01	0.4	0.46	0.44	1.00
3	16	0.01	0.5	0.46	0.44	1.00
3	32	0.00001	0.2	1.52	2.74	2.50
3	32	0.00001	0.3	1.36	2.27	2.28
3	32	0.00001	0.4	0.78	0.90	1.44
3	32	0.00001	0.5	0.51	0.49	1.06
3	32	0.0001	0.2	1.60	3.01	2.62
3	32	0.0001	0.3	1.63	3.11	2.67
3	32	0.0001	0.4	1.58	2.93	2.59
3	32	0.0001	0.5	1.52	2.74	2.50
3	32	0.001	0.2	0.46	0.43	0.99
3	32	0.001	0.3	0.46	0.44	1.00
3	32	0.001	0.4	0.47	0.44	1.00
3	32	0.001	0.5	0.48	0.45	1.01
3	32	0.01	0.2	0.46	0.44	1.00
3	32	0.01	0.3	0.46	0.44	1.00
3	32	0.01	0.4	0.46	0.44	1.00
3	32	0.01	0.5	0.46	0.44	1.00

Table M.4: Validation performance metrics for longitude using the Dynamic Edges GNN Architecture on reduced dataset (part 1).

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
3	64	0.00001	0.2	0.95	1.25	1.69
3	64	0.00001	0.3	0.90	1.11	1.59
3	64	0.00001	0.4	0.92	1.12	1.60
3	64	0.00001	0.5	0.71	0.77	1.33
3	64	0.0001	0.2	1.08	1.50	1.85
3	64	0.0001	0.3	1.62	3.05	2.64
3	64	0.0001	0.4	1.58	2.94	2.59
3	64	0.0001	0.5	1.37	2.27	2.28
3	64	0.001	0.2	0.48	0.49	1.06
3	64	0.001	0.3	0.47	0.44	1.00
3	64	0.001	0.4	0.48	0.47	1.04
3	64	0.001	0.5	1.62	3.06	2.65
3	64	0.01	0.2	0.48	0.47	1.04
3	64	0.01	0.3	0.45	0.44	1.00
3	64	0.01	0.4	0.46	0.45	1.01
3	64	0.01	0.5	0.47	0.44	1.00
4	16	0.00001	0.2	1.30	2.10	2.19
4	16	0.00001	0.3	1.28	2.04	2.16
4	16	0.00001	0.4	1.45	2.51	2.40
4	16	0.00001	0.5	0.50	0.45	1.01
4	16	0.0001	0.2	0.48	0.47	1.04
4	16	0.0001	0.3	1.61	3.04	2.64
4	16	0.0001	0.4	1.62	3.07	2.65
4	16	0.0001	0.5	1.48	2.64	2.46
4	16	0.001	0.2	0.46	0.44	1.00
4	16	0.001	0.3	0.44	0.44	1.00
4	16	0.001	0.4	0.47	0.45	1.01
4	16	0.001	0.5	0.47	0.45	1.01
4	16	0.01	0.2	0.46	0.44	1.00
4	16	0.01	0.3	0.46	0.44	1.00
4	16	0.01	0.4	0.46	0.44	1.00
4	16	0.01	0.5	0.46	0.44	1.00
4	32	0.00001	0.2	1.47	2.60	2.44
4	32	0.00001	0.3	1.50	2.67	2.47
4	32	0.00001	0.4	1.08	1.52	1.87
4	32	0.00001	0.5	0.70	0.76	1.32
4	32	0.0001	0.2	0.50	0.50	1.07
4	32	0.0001	0.3	1.61	3.03	2.63
4	32	0.0001	0.4	1.61	3.04	2.64
4	32	0.0001	0.5	1.48	2.65	2.46
4	32	0.001	0.2	1.99	4.78	3.31
4	32	0.001	0.3	0.46	0.44	1.00
4	32	0.001	0.4	0.47	0.44	1.00
4	32	0.001	0.5	1.62	3.07	2.65
4	32	0.01	0.2	0.48	0.46	1.03
4	32	0.01	0.3	0.47	0.44	1.00
4	32	0.01	0.4	0.46	0.44	1.00
4	32	0.01	0.5	0.46	0.44	1.00

Table M.5: Validation performance metrics for longitude using the Dynamic Edges GNN Architecture on reduced dataset (part 2).

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
4	64	0.00001	0.2	1.56	2.85	2.55
4	64	0.00001	0.3	1.30	2.13	2.21
4	64	0.00001	0.4	0.97	1.27	1.70
4	64	0.00001	0.5	0.65	0.65	1.22
4	64	0.0001	0.2	1.64	3.12	2.67
4	64	0.0001	0.3	1.58	2.92	2.59
4	64	0.0001	0.4	1.64	3.11	2.67
4	64	0.0001	0.5	1.18	1.79	2.02
4	64	0.001	0.2	0.46	0.44	1.00
4	64	0.001	0.3	1.64	3.13	2.68
4	64	0.001	0.4	1.64	3.14	2.68
4	64	0.001	0.5	1.64	3.13	2.68
4	64	0.01	0.2	0.46	0.45	1.01
4	64	0.01	0.3	0.46	0.44	1.00
4	64	0.01	0.4	0.47	0.46	1.03
4	64	0.01	0.5	0.48	0.45	1.01

Table M.6: Validation performance metrics for longitude using the Dynamic Edges GNN Architecture on reduced dataset (part 3).

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
3	16	0.00001	0.2	10.81	172.07	1.62
3	16	0.00001	0.3	12.07	204.80	1.77
3	16	0.00001	0.4	13.71	253.72	1.97
3	16	0.00001	0.5	7.50	75.14	1.07
3	16	0.0001	0.2	6.57	58.11	0.94
3	16	0.0001	0.3	13.81	255.92	1.97
3	16	0.0001	0.4	10.63	166.48	1.59
3	16	0.0001	0.5	15.11	295.16	2.12
3	16	0.001	0.2	7.73	90.10	1.17
3	16	0.001	0.3	7.07	79.39	1.10
3	16	0.001	0.4	7.36	83.67	1.13
3	16	0.001	0.5	7.03	63.11	0.98
3	16	0.01	0.2	7.39	69.39	1.03
3	16	0.01	0.3	7.36	68.65	1.02
3	16	0.01	0.4	7.36	68.67	1.02
3	16	0.01	0.5	7.37	69.15	1.03
3	32	0.00001	0.2	7.25	66.31	1.01
3	32	0.00001	0.3	7.39	69.31	1.03
3	32	0.00001	0.4	12.83	28.69	0.66
3	32	0.00001	0.5	8.93	125.28	1.38
3	32	0.0001	0.2	6.96	62.19	0.97
3	32	0.0001	0.3	14.34	267.27	2.02
3	32	0.0001	0.4	7.53	77.03	1.08
3	32	0.0001	0.5	9.03	122.98	1.37
3	32	0.001	0.2	6.61	60.78	0.96
3	32	0.001	0.3	6.77	59.68	0.95
3	32	0.001	0.4	6.93	59.89	0.96
3	32	0.001	0.5	7.28	66.82	1.01
3	32	0.01	0.2	7.11	65.90	1.00
3	32	0.01	0.3	7.36	68.65	1.02
3	32	0.01	0.4	7.35	68.03	1.02
3	32	0.01	0.5	7.37	68.77	1.02

Table M.7: Validation performance metrics for depth using the Dynamic Edges GNN Architecture on reduced dataset (part 1).

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
3	64	0.00001	0.2	7.30	68.45	1.02
3	64	0.00001	0.3	7.55	84.22	1.13
3	64	0.00001	0.4	8.19	96.17	1.21
3	64	0.00001	0.5	8.05	97.37	1.22
3	64	0.0001	0.2	7.46	77.78	1.09
3	64	0.0001	0.3	7.62	85.01	1.14
3	64	0.0001	0.4	8.28	104.74	1.26
3	64	0.0001	0.5	8.46	110.98	1.30
3	64	0.001	0.2	7.62	88.19	1.16
3	64	0.001	0.3	7.20	65.43	1.00
3	64	0.001	0.4	7.14	64.06	0.99
3	64	0.001	0.5	15.14	295.00	2.12
3	64	0.01	0.2	7.31	82.86	1.12
3	64	0.01	0.3	6.91	73.22	1.06
3	64	0.01	0.4	6.83	72.64	1.05
3	64	0.01	0.5	6.52	56.61	0.93
4	16	0.00001	0.2	7.10	64.18	0.99
4	16	0.00001	0.3	8.62	114.03	1.32
4	16	0.00001	0.4	9.72	142.47	1.47
4	16	0.00001	0.5	7.44	70.59	1.04
4	16	0.0001	0.2	7.06	62.65	0.98
4	16	0.0001	0.3	13.71	252.75	1.96
4	16	0.0001	0.4	12.58	222.71	1.84
4	16	0.0001	0.5	14.17	264.06	2.01
4	16	0.001	0.2	7.02	71.13	1.04
4	16	0.001	0.3	6.51	70.14	1.03
4	16	0.001	0.4	6.28	54.04	0.91
4	16	0.001	0.5	6.75	57.08	0.93
4	16	0.01	0.2	7.35	68.29	1.02
4	16	0.01	0.3	7.35	68.31	1.02
4	16	0.01	0.4	7.34	67.92	1.02
4	16	0.01	0.5	7.34	67.95	1.02
4	32	0.00001	0.2	7.22	66.98	1.01
4	32	0.00001	0.3	9.64	141.57	1.47
4	32	0.00001	0.4	7.45	70.67	1.04
4	32	0.00001	0.5	7.20	65.32	1.00
4	32	0.0001	0.2	7.05	62.28	0.97
4	32	0.0001	0.3	7.56	76.59	1.08
4	32	0.0001	0.4	7.34	58.19	0.94
4	32	0.0001	0.5	8.02	99.44	1.23
4	32	0.001	0.2	15.93	307.50	2.16
4	32	0.001	0.3	6.50	63.72	0.99
4	32	0.001	0.4	7.29	66.96	1.01
4	32	0.001	0.5	15.64	310.13	2.17
4	32	0.01	0.2	6.66	63.73	0.99
4	32	0.01	0.3	7.08	62.94	0.98
4	32	0.01	0.4	7.35	68.06	1.02
4	32	0.01	0.5	7.35	68.27	1.02

Table M.8: Validation performance metrics for depth using the Dynamic Edges GNN Architecture on reduced dataset (part 2).

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
4	64	0.00001	0.2	9.82	148.48	1.50
4	64	0.00001	0.3	7.68	85.14	1.14
4	64	0.00001	0.4	7.43	78.51	1.09
4	64	0.00001	0.5	9.89	149.88	1.51
4	64	0.0001	0.2	7.75	89.29	1.17
4	64	0.0001	0.3	8.77	115.73	1.33
4	64	0.0001	0.4	7.61	84.16	1.13
4	64	0.0001	0.5	7.81	89.81	1.17
4	64	0.001	0.2	7.06	62.71	0.98
4	64	0.001	0.3	15.79	315.02	2.19
4	64	0.001	0.4	14.03	262.09	2.00
4	64	0.001	0.5	15.77	314.44	2.19
4	64	0.01	0.2	6.97	72.03	1.05
4	64	0.01	0.3	6.46	62.73	0.98
4	64	0.01	0.4	6.76	67.34	1.01
4	64	0.01	0.5	6.61	55.54	0.92

Table M.9: Validation performance metrics for depth using the Dynamic Edges GNN Architecture on reduced dataset (part 3).

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
3	16	0.00001	0.2	0.34	0.24	1.43
3	16	0.00001	0.3	0.34	0.23	1.40
3	16	0.00001	0.4	0.33	0.22	1.37
3	16	0.00001	0.5	0.30	0.18	1.24
3	16	0.0001	0.2	0.35	0.24	1.43
3	16	0.0001	0.3	0.35	0.24	1.43
3	16	0.0001	0.4	0.34	0.24	1.43
3	16	0.0001	0.5	0.34	0.24	1.43
3	16	0.001	0.2	0.32	0.16	1.17
3	16	0.001	0.3	0.27	0.13	1.05
3	16	0.001	0.4	0.27	0.14	1.09
3	16	0.001	0.5	0.27	0.14	1.09
3	16	0.01	0.2	0.27	0.14	1.09
3	16	0.01	0.3	0.27	0.14	1.09
3	16	0.01	0.4	0.27	0.14	1.09
3	16	0.01	0.5	0.27	0.14	1.09
3	32	0.00001	0.2	0.34	0.23	1.40
3	32	0.00001	0.3	0.34	0.23	1.40
3	32	0.00001	0.4	0.32	0.21	1.34
3	32	0.00001	0.5	0.29	0.15	1.13
3	32	0.0001	0.2	0.35	0.24	1.43
3	32	0.0001	0.3	0.35	0.24	1.43
3	32	0.0001	0.4	0.34	0.24	1.43
3	32	0.0001	0.5	0.34	0.23	1.40
3	32	0.001	0.2	0.28	0.14	1.09
3	32	0.001	0.3	0.27	0.14	1.09
3	32	0.001	0.4	0.27	0.14	1.09
3	32	0.001	0.5	0.27	0.13	1.05
3	32	0.01	0.2	0.27	0.14	1.09
3	32	0.01	0.3	0.27	0.14	1.09
3	32	0.01	0.4	0.27	0.14	1.09
3	32	0.01	0.5	0.27	0.14	1.09

Table M.10: Validation performance metrics for magnitude using the Dynamic Edges GNN Architecture on reduced dataset (part 1).

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
3	64	0.00001	0.2	0.34	0.23	1.40
3	64	0.00001	0.3	0.34	0.23	1.40
3	64	0.00001	0.4	0.30	0.18	1.24
3	64	0.00001	0.5	0.28	0.13	1.05
3	64	0.0001	0.2	0.34	0.24	1.43
3	64	0.0001	0.3	0.34	0.24	1.43
3	64	0.0001	0.4	0.34	0.23	1.40
3	64	0.0001	0.5	0.34	0.23	1.40
3	64	0.001	0.2	0.28	0.14	1.09
3	64	0.001	0.3	0.27	0.14	1.09
3	64	0.001	0.4	0.27	0.13	1.05
3	64	0.001	0.5	0.34	0.24	1.43
3	64	0.01	0.2	0.27	0.14	1.09
3	64	0.01	0.3	0.27	0.14	1.09
3	64	0.01	0.4	0.27	0.14	1.09
3	64	0.01	0.5	0.27	0.14	1.09
4	16	0.00001	0.2	0.34	0.23	1.40
4	16	0.00001	0.3	0.34	0.23	1.40
4	16	0.00001	0.4	0.34	0.23	1.40
4	16	0.00001	0.5	0.28	0.15	1.13
4	16	0.0001	0.2	0.27	0.14	1.09
4	16	0.0001	0.3	0.35	0.24	1.43
4	16	0.0001	0.4	0.34	0.24	1.43
4	16	0.0001	0.5	0.34	0.24	1.43
4	16	0.001	0.2	0.27	0.13	1.05
4	16	0.001	0.3	0.27	0.14	1.09
4	16	0.001	0.4	0.27	0.14	1.09
4	16	0.001	0.5	0.27	0.14	1.09
4	16	0.01	0.2	0.27	0.14	1.09
4	16	0.01	0.3	0.27	0.14	1.09
4	16	0.01	0.4	0.27	0.14	1.09
4	16	0.01	0.5	0.27	0.14	1.09
4	32	0.00001	0.2	0.34	0.23	1.40
4	32	0.00001	0.3	0.34	0.23	1.40
4	32	0.00001	0.4	0.33	0.22	1.37
4	32	0.00001	0.5	0.29	0.17	1.20
4	32	0.0001	0.2	0.28	0.15	1.13
4	32	0.0001	0.3	0.34	0.24	1.43
4	32	0.0001	0.4	0.34	0.23	1.40
4	32	0.0001	0.5	0.34	0.23	1.40
4	32	0.001	0.2	0.37	0.26	1.49
4	32	0.001	0.3	0.27	0.13	1.05
4	32	0.001	0.4	0.27	0.13	1.05
4	32	0.001	0.5	0.34	0.24	1.43
4	32	0.01	0.2	0.27	0.14	1.09
4	32	0.01	0.3	0.27	0.14	1.09
4	32	0.01	0.4	0.27	0.14	1.09
4	32	0.01	0.5	0.27	0.14	1.09

Table M.11: Validation performance metrics for magnitude using the Dynamic Edges GNN Architecture on reduced dataset (part 2).

k	Batch Size	LR	Dropout Prob	MAE	MSE	NRMSE
4	64	0.00001	0.2	0.34	0.23	1.40
4	64	0.00001	0.3	0.32	0.21	1.34
4	64	0.00001	0.4	0.29	0.16	1.17
4	64	0.00001	0.5	0.40	0.20	1.30
4	64	0.0001	0.2	0.34	0.24	1.43
4	64	0.0001	0.3	0.34	0.24	1.43
4	64	0.0001	0.4	0.34	0.24	1.43
4	64	0.0001	0.5	0.34	0.23	1.40
4	64	0.001	0.2	0.27	0.13	1.05
4	64	0.001	0.3	0.35	0.24	1.43
4	64	0.001	0.4	0.35	0.24	1.43
4	64	0.001	0.5	0.35	0.24	1.43
4	64	0.01	0.2	0.27	0.14	1.09
4	64	0.01	0.3	0.27	0.14	1.09
4	64	0.01	0.4	0.27	0.14	1.09
4	64	0.01	0.5	0.27	0.14	1.09

Table M.12: Validation performance metrics for magnitude using the Dynamic Edges GNN Architecture on reduced dataset (part 3).

k	Batch Size	LR	Dropout Prob	NRMSE
3	16	0.00001	0.2	2.00
3	16	0.00001	0.3	1.95
3	16	0.00001	0.4	1.54
3	16	0.00001	0.5	1.43
3	16	0.0001	0.2	1.85
3	16	0.0001	0.3	2.10
3	16	0.0001	0.4	2.00
3	16	0.0001	0.5	2.12
3	16	0.001	0.2	1.17
3	16	0.001	0.3	1.17
3	16	0.001	0.4	1.06
3	16	0.001	0.5	1.03
3	16	0.01	0.2	1.04
3	16	0.01	0.3	1.04
3	16	0.01	0.4	1.04
3	16	0.01	0.5	1.03
3	32	0.00001	0.2	1.78
3	32	0.00001	0.3	1.70
3	32	0.00001	0.4	1.28
3	32	0.00001	0.5	1.24
3	32	0.0001	0.2	1.85
3	32	0.0001	0.3	2.12
3	32	0.0001	0.4	1.86
3	32	0.0001	0.5	1.89
3	32	0.001	0.2	1.01
3	32	0.001	0.3	1.02
3	32	0.001	0.4	1.02
3	32	0.001	0.5	1.04
3	32	0.01	0.2	1.03
3	32	0.01	0.3	1.04
3	32	0.01	0.4	1.04
3	32	0.01	0.5	1.03

Table M.13: Validation performance metrics for the four target variables using the Dynamic Edges GNN Architecture on reduced dataset (part 1).

k	Batch Size	LR	Dropout Prob	NRMSE
3	64	0.00001	0.2	1.56
3	64	0.00001	0.3	1.45
3	64	0.00001	0.4	1.27
3	64	0.00001	0.5	1.22
3	64	0.0001	0.2	1.62
3	64	0.0001	0.3	1.89
3	64	0.0001	0.4	1.89
3	64	0.0001	0.5	1.77
3	64	0.001	0.2	1.12
3	64	0.001	0.3	1.03
3	64	0.001	0.4	1.03
3	64	0.001	0.5	2.14
3	64	0.01	0.2	1.09
3	64	0.01	0.3	1.05
3	64	0.01	0.4	1.06
3	64	0.01	0.5	1.02
4	16	0.00001	0.2	1.68
4	16	0.00001	0.3	1.77
4	16	0.00001	0.4	1.81
4	16	0.00001	0.5	1.04
4	16	0.0001	0.2	1.03
4	16	0.0001	0.3	2.10
4	16	0.0001	0.4	2.07
4	16	0.0001	0.5	2.06
4	16	0.001	0.2	1.03
4	16	0.001	0.3	1.04
4	16	0.001	0.4	1.01
4	16	0.001	0.5	1.01
4	16	0.01	0.2	1.04
4	16	0.01	0.3	1.04
4	16	0.01	0.4	1.04
4	16	0.01	0.5	1.04
4	32	0.00001	0.2	1.77
4	32	0.00001	0.3	1.88
4	32	0.00001	0.4	1.48
4	32	0.00001	0.5	1.18
4	32	0.0001	0.2	1.04
4	32	0.0001	0.3	1.87
4	32	0.0001	0.4	1.82
4	32	0.0001	0.5	1.83
4	32	0.001	0.2	3.00
4	32	0.001	0.3	1.01
4	32	0.001	0.4	1.02
4	32	0.001	0.5	2.15
4	32	0.01	0.2	1.03
4	32	0.01	0.3	1.03
4	32	0.01	0.4	1.04
4	32	0.01	0.5	1.04

Table M.14: Validation performance metrics for the four target variables using the Dynamic Edges GNN Architecture on reduced dataset (part 2).

k	Batch Size	LR	Dropout Prob	NRMSE
4	64	0.00001	0.2	1.91
4	64	0.00001	0.3	1.72
4	64	0.00001	0.4	1.56
4	64	0.00001	0.5	1.28
4	64	0.0001	0.2	1.91
4	64	0.0001	0.3	1.92
4	64	0.0001	0.4	1.88
4	64	0.0001	0.5	1.71
4	64	0.001	0.2	1.01
4	64	0.001	0.3	2.15
4	64	0.001	0.4	2.12
4	64	0.001	0.5	2.16
4	64	0.01	0.2	1.05
4	64	0.01	0.3	1.05
4	64	0.01	0.4	1.04
4	64	0.01	0.5	1.01

Table M.15: Validation performance metrics for the four target variables using the Dynamic Edges GNN Architecture on reduced dataset (part 3).

References

Babai, L., & Kucera, L. (1979). Canonical labelling of graphs in linear average time. In *20th annual symposium on foundations of computer science (sfcs 1979)* (p. 39-46).

Banach, S. (1929a). Sur les fonctionnelles linéaires. *Studia Mathematica*, *1*, 211–216.

Banach, S. (1929b). Sur les fonctionnelles linéaires ii. *Studia Mathematica*, *1*, 223–239.

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In G. Montavon, G. B. Orr, & K.-R. Müller (Eds.), *Neural networks: Tricks of the trade: Second edition* (pp. 437–478). Berlin, Heidelberg: Springer Berlin Heidelberg.

Beskardes, G. D., Hole, J. A., Wang, K., Michaelides, M., Wu, Q., Chapman, M. C., ... Quiros, D. A. (2017, 12). A comparison of earthquake backprojection imaging methods for dense local arrays. *Geophysical Journal International*, *212*(3), 1986-2002.

Beyreuther, M., Barsch, R., Krischer, L., Megies, T., Behr, Y., & Wassermann, J. (2010, 06). Obspy: A python toolbox for seismology. *Seismological Research Letters*, *81*, 530-533. doi: 10.1785/gssrl.81.3.530

Bezdek, J., & Pal, N. (1998). Some new indexes of cluster validity. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *28*(3), 301-315.

Bladford, R. (1993). *Discrimination of earthquakes and explosions* (Technical Report No. AFTAC-TR-93-044 HQ). Patrick Air Force Base, FL: Air Force Technical Applications Center. Retrieved from https://apps.dtic.mil/sti/pdfs/ADA267638.pdf

Blashfield, R. K. (1976). Mixture model tests of cluster analysis: Accuracy of four agglomerative hierarchical methods. *Psychological Bulletin*, *83*, 377–388.

Bottou, L. (2012). Stochastic gradient descent tricks. In G. Montavon, G. B. Orr, & K.-R. Müller (Eds.), *Neural networks: Tricks of the trade: Second edition* (pp. 421–436). Berlin, Heidelberg: Springer Berlin Heidelberg.

Campello, R. J. G. B., Moulavi, D., & Sander, J. (2013). Density-based clustering based on hierarchical density estimates. In J. Pei, V. S. Tseng, L. Cao, H. Motoda, & G. Xu (Eds.), *Advances in knowledge discovery and data mining* (pp. 160–172). Berlin, Heidelberg: Springer Berlin Heidelberg.

Chan, A. (1982). *The glory and fall of the ming dynasty.* Norman: University of Oklahoma Press.

Chollet, F. (2017, July). Xception: Deep Learning with Depthwise Separable Convolutions . In *2017 ieee conference on computer vision and pattern recognition (cvpr)* (p. 1800-1807). Los Alamitos, CA, USA: IEEE Computer Society.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, *2*, 303-314.

Du, J. (2019, may). The frontier of sgd and its variants in machine learning. *Journal of Physics: Conference Series*, *1229*(1), 012046.

Dunn, J. C. (1973). A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, *3*(3), 32–57.

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Knowledge discovery and data mining.* Retrieved from https://api.semanticscholar.org/CorpusID:355163

European Environment Agency. (2009). *Mediterranean Sea Physiography.* Retrieved from https://www.eea.europa.eu/data-and-maps/figures/mediterranean-sea-physiography/figure-01-1pia.eps

Frasconi, P., Gori, M., & Sperduti, A. (1998). A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks*, *9*(5), 768-786. doi: 10.1109/72.712151

Friedman, H. P., & Rubin, J. (1967). On some invariant criteria for grouping data. *Journal of the American Statistical Association*, *62*(320), 1159–1178.

Fukushima, K. (1980, April). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, *36*(4), 193–202. Retrieved from https://doi.org/10.1007/BF00344251 doi: 10.1007/BF00344251

Gajewski, D., Anikiev, D., Kashtan, B., & Tessmer, E. (2007). Localization of seismic events by diffraction stacking. *SEG technical program expanded abstracts 2007*, 1287-1291.

Gal, Y., & Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In M. F. Balcan & K. Q. Weinberger (Eds.), *Proceedings of the 33rd international conference on machine learning* (Vol. 48, pp. 1050–1059). PMLR. Retrieved from https://proceedings.mlr.press/v48/gal16.html

Gere, J., & Shah, H. (1980). *The 1976 tangshan, china earthquake: Papers presented at the 2nd u.s. national conference on earthquake engineering held at stanford university, august 22–24, 1979.* Berkeley, California: Earthquake Engineering Research Center.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017, 06–11 Aug). Neural message passing for quantum chemistry. In D. Precup & Y. W. Teh (Eds.), *Proceedings of the 34th international conference on machine learning* (Vol. 70, pp. 1263–1272). PMLR.

Glorot, X., & Bengio, Y. (2010, 13–15 May). Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh & M. Titterington (Eds.), *Proceedings of the thirteenth*

international conference on artificial intelligence and statistics (Vol. 9, pp. 249–256). Chia Laguna Resort, Sardinia, Italy: PMLR.

Goodfellow, I. J., Bengio, Y., & Courville, A. (2016). *Deep learning.* Cambridge, MA, USA: MIT Press. (http://www.deeplearningbook.org)

Gori, M., Monfardini, G., & Scarselli, F. (2005). A new model for learning in graph domains. In *Proceedings. 2005 ieee international joint conference on neural networks, 2005.* (Vol. 2, p. 729-734). doi: 10.1109/IJCNN.2005.1555942

Gower, J. C. (1971). A general coefficient of similarity and some of its properties. *Biometrics*, *27*(4), 857–871.

Grover, A., & Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (p. 855–864). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2939672.2939754 doi: 10.1145/2939672.2939754

Hahn, H. (1927). Über lineare gleichungssysteme in linearen räumen. *Journal für die reine und angewandte Mathematik*, *157*, 214–229.

Hamilton, W. L. (2020). *Graph representation learning* (1st ed.). Springer Cham.

Han, J., Kamber, M., & Pei, J. (2012). *Data mining: Concepts and techniques* (3rd ed.). Waltham, MA: Morgan Kaufmann Publishers.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.* Retrieved from https://arxiv.org/abs/1502.01852

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *2016 ieee conference on computer vision and pattern recognition (cvpr)* (p. 770-778). doi: 10.1109/CVPR.2016.90

Hopkins, B., & Skellam, J. G. (1954, 04). A new method for determining the type of distribution of plant individuals. *Annals of Botany*, *18*(2), 213-227.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, *4*(2), 251-257.

Huber, P. J. (1992). Robust estimation of a location parameter. In S. Kotz & N. L. Johnson (Eds.), *Breakthroughs in statistics: Methodology and distribution* (pp. 492–518). New York, NY: Springer New York.

Ilc, N. (2012). Modified dunn's cluster validity index based on graph theory. *Przeglad Elektrotechniczny*, *88*(2), 126–131. (In English)

Institut de Ciències del Mar (CSIC). (2021). *"After the Ring of Fire, the Mediterranean is one of the most active seismic zones" – Interview with Ferran Estrada.* https://www.icm.csic.es/en/news/after-ring-fire-mediterranean-one-most-active-seismic-zones-interview-ferran-estrada. (Online; accessed 23 June 2025)

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR*, *abs/1502.03167*.

Jain, A. (2024). *Pooling and their types in CNN.* Retrieved from https://medium.com/@abhishekjainindore24/pooling-and-their-types-in-cnn-4a4b8a7a4611

Johnson, D. B. (1977, January). Efficient algorithms for shortest paths in sparse networks. *J. ACM*, *24*(1), 1–13.

Jordan, J. (2018). *Setting the learning rate of your neural networks.* Retrieved from https://www.jeremyjordan.me/nn-learning-rate/

Kakutani, S. (1941). Concrete representation of abstract (m)-spaces. (a characterization of the space of continuous functions.). *Annals of Mathematics*, *42*(4), 994–1024.

Karthik, R., Menaka, R., Kathiresan, G. S., Anirudh, M., & Nagharjun, M. (2022). Gaussian dropout based stacked ensemble cnn for classification of breast tumor in ultrasound images. *IRBM*, *43*(6), 715–733. Retrieved from https://doi.org/10.1016/j.irbm.2022.03.005 doi: 10.1016/j.irbm.2022.03.005

Keras Team. (2025). *Batchnormalization layer documentation.* https://keras.io/api/layers/normalization_layers/batch_normalization/. (Accessed: 12-04-2025)

Kiefer, J., & Wolfowitz, J. (1952). Stochastic Estimation of the Maximum of a Regression Function. *The Annals of Mathematical Statistics*, *23*(3), 462 – 466.

Kingma, D., & Ba, J. (2017). Adam: A method for stochastic optimization. In *3rd internation conference on learning representations 2015* (p. 1-15).

Kriegerowski, M., Petersen, G. M., Vasyura-Bathke, H., & Ohrnberger, M. (2018, 12). A Deep Convolutional Neural Network for Localization of Clustered Earthquakes Based on Multistation Full Waveforms. *Seismological Research Letters*, *90*(2A), 510-516.

Krizhevsky, A., Sutskever, I., & Hinton, G. (2012, 01). Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, *25*. doi: 10.1145/3065386

Kumar, A. (2023). *Different types of cnn architectures explained (with examples).* https://vitalflux.com/different-types-of-cnn-architectures-explained-examples/.

Lawson, R. G., & Jurs, P. C. (1990). New index for clustering tendency and its application to chemical problems. *Journal of Chemical Information and Computer Sciences*, *30*(1), 36–41. Retrieved from https://doi.org/10.1021/ci00065a010 doi: 10.1021/ci00065a010

Lay, T., Kanamori, H., Ammon, C. J., Nettles, M., Ward, S. N., Aster, R. C., ... Sipkin, S. A. (2005, May). The great sumatra-andaman earthquake of 26 december 2004. *Science*, *308*(5725), 1127–1133. Retrieved from https://www.science.org/doi/10.1126/science.1112250 doi: 10.1126/science.1112250

LeCun, Y. (1989). Generalization and network design strategies. In R. Pfeifer, Z. Schreter, F. Fogelman, & L. Steels (Eds.), *Connectionism in perspective.* Elsevier.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324. doi: 10.1109/5.726791

Lee, C.-Y., Gallagher, P., & Tu, Z. (2017). Generalizing pooling functions in cnns: Mixed, gated, and tree. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PP*.

Leshno, M., Lin, V. Y., Pinkus, A., & Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, *6*(6), 861-867.

Leskovec, J., Rajaraman, A., & Ullman, J. D. (2014). *Mining of massive datasets.* Cambridge University Press.

Li, L., Tan, J., Schwarz, B., Staněk, F., Poiata, N., Shi, P., ... Gajewski, D. (2020). Recent advances and challenges of waveform-based seismic location methods at multiple scales. *Reviews of Geophysics*, *58*(1), e2019RG000667.

Li, Z., Meier, M.-A., Hauksson, E., Zhan, Z., & Andrews, J. (2018, 05). Machine learning seismic wave discrimination: Application to earthquake early warning. *Geophysical Research Letters*, *45*.

Li, Z., & Van der Baan, M. (2016). Microseismic event localization by acoustic time reversal extrapolation. *Geophysics*, *81*, KS123-KS134.

Lin, Y., Syracuse, E. M., Maceira, M., Zhang, H., & Larmat, C. (2015, 03). Double-difference traveltime tomography with edge-preserving regularization and a priori interfaces. *Geophysical Journal International*, *201*, 574-594.

Loshchilov, I., & Hutter, F. (2019). *Decoupled weight decay regularization.* Retrieved from https://arxiv.org/abs/1711.05101

Lutzeyer, J. F., Wu, C., & Vazirgiannis, M. (2022). Sparsifying the update step in graph neural networks. In *Topological, algebraic and geometric learning workshops 2022* (pp. 258–268).

Ma, J., Bunge, H.-P., Thrastarson, S., Fichtner, A., Herwaarden, D.-P. v., Tian, Y., ... Liu, T. (2022). Seismic full-waveform inversion of the crust-mantle structure beneath china and adjacent regions. *Journal of Geophysical Research: Solid Earth*, *127*(9).

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In L. M. L. Cam & J. Neyman (Eds.), *Proceedings of the fifth berkeley symposium on mathematical statistics and probability* (Vol. 1, pp. 281–297). Berkeley, CA: University of California Press.

Markov, A. (1938). On mean values and exterior densities. *Recueil Mathématique de Moscou, Nouvelle Série*, *4*, 165–190.

McBrearty, I. W., & Beroza, G. C. (2022). Earthquake location and magnitude estimation with graph neural networks. In *2022 ieee international conference on image processing (icip)* (p. 3858-3862).

McCulloch, W., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics.*

Micheli, A. (2009, Mar). Neural network for graphs: a contextual constructive approach. *IEEE Transactions on Neural Networks*, *20*(3), 498–511. doi: 10.1109/TNN.2008.2010350

Minsky, M., & Papert, S. (1969). *Perceptrons.* M.I.T Press.

Mitchell, T. (1997). *Maching learning.* McGraw-Hill.

Morris, C., Lipman, Y., Maron, H., Rieck, B., Kriege, N. M., Grohe, M., ... Borgwardt, K. (2023). Weisfeiler and leman go machine learning: The story so far. *Journal of Machine Learning Research*, *24*(333), 1–59.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., & Grohe, M. (2019). Weisfeiler and leman go neural: higher-order graph neural networks. In *Proceedings of the thirty-third aaai conference on artificial intelligence and thirty-first innovative applications of artificial intelligence conference and ninth aaai symposium on educational advances in artificial intelligence.* AAAI Press.

Mousavi, S. M., & Beroza, G. C. (2020a). Bayesian-deep-learning estimation of earthquake location from single-station observations. *IEEE Transactions on Geoscience and Remote Sensing*, *58*, 8211-8224.

Mousavi, S. M., & Beroza, G. C. (2020b). A machine-learning approach for earthquake magnitude estimation. *Geophysical Research Letters*, *47*.

Münchmeyer, J., Bindi, D., Leser, U., & Tilmann, F. (2020, 12). The transformer earthquake alerting model: a new versatile approach to earthquake early warning. *Geophysical Journal International*, *225*(1), 646-656.

Münchmeyer, J., Bindi, D., Leser, U., & Tilmann, F. (2021, 04). Earthquake magnitude and location estimation from real time seismic waveforms with a transformer network. *Geophysical Journal International*, *226*(2), 1086-1104.

National Oceanic and Atmospheric Administration. (1960). *May 22, 1960 southern chile earthquake and tsunami.* (U.S. Department of Commerce. PDF document)

Ng, A. Y. (2004). Feature selection, l1 vs. l2 regularization, and rotational invariance. In (p. 78). New York, NY, USA: Association for Computing Machinery.

Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, S., ... Snoek, J. (2019). Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *Advances in neural information processing systems* (Vol. 32).

Pal, N., & Biswas, J. (1997). Cluster validation using graph theoretic concepts. *Pattern Recognition*, *30*(6), 847-857.

Perol, T., Gharbi, M., & Denolle, M. (2018). Convolutional neural network for earthquake detection and location. *Science Advances*, *4*.

Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). Deepwalk: online learning of social representations. In *Proceedings of the 20th acm sigkdd international conference on knowledge discovery and data mining* (p. 701–710). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2623330.2623732 doi: 10.1145/2623330.2623732

Pesicek, J. D., Child, D., Artman, B., & Cieślik, K. (2014). Picking versus stacking in a modern microearthquake location: Comparison of results from a surface passive seismic monitoring array in Oklahoma. *Geophysics*, *79*, KS61-KS68.

PyTorch Contributors. (2024). *Batchnorm2d documentation.* https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html. (Accessed: 12-04-2025)

Qi, C. R., Yi, L., Su, H., & Guibas, L. J. (2017). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In I. Guyon et al. (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc.

Radford, A., Metz, L., & Chintala, S. (2016). *Unsupervised representation learning with deep convolutional generative adversarial networks.* Retrieved from https://arxiv.org/abs/1511.06434

Riesz, F. (1909). Sur les opérations fonctionnelles linéaires. *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences*, *149*, 974–977.

Robbins, H., & Monro, S. (1951). A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, *22*(3), 400 – 407.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *The perceptron: A probabilistic model for information storage and organization in the brain.*.

Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, *20*, 53-65.

Rumelhart, D., Durbin, R., Golden, R., & Chauvin, Y. (1995). Chapter 1: Backpropogation: The basic theory. In D. E. R. Yves Chauvin (Ed.), *Backpropagation: Theory, Architectures, and Applications* (p. 1-34). Psychology Press.

Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, *323*, 533-536. Retrieved from https://api.semanticscholar.org/CorpusID:205001834

Sanchez-Lengeling, B., Reif, E., Pearce, A., & Wiltschko, A. B. (2021). A gentle introduction to graph neural networks. *Distill*. (https://distill.pub/2021/gnn-intro) doi: 10.23915/distill.00033

Sander, J., Ester, M., Kriegel, H.-P., & Xu, X. (1998, June). Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data Mining and Knowledge Discovery*, *2*(2), 169–194.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. *IEEE transactions on neural networks*, *20*(1), 61–80.

Sengupta, U., Amos, M., Hosking, J. S., Rasmussen, C. E., Juniper, M. P., & Young, P. J. (2020). Ensembling geophysical models with bayesian neural networks. In *Proceedings of the 34th international conference on neural information processing systems.* Red Hook, NY, USA: Curran Associates Inc.

Sharma, M. (2024, 11). Deep learning-based classification of seismic events using waveform data. *Advances and Applications in Discrete Mathematics*, *42*, 17-45. doi: 10.17654/0974165825002

Shen, H., & Shen, Y. (2021, 04). Array-Based Convolutional Neural Networks for Automatic Detection and 4D Localization of Earthquakes in Hawai'i. *Seismological Research Letters*, *92*(5), 2961-2971.

Shin, S., Hoàng, T., Le, T.-H., & Lee, M.-Y. (2016, 02). A new robust design method using neural network. *Journal of Nanoelectronics and Optoelectronics*, *11*, 68-78.

Shrivastava, A. (2022). *Lecture 6: Deep learning.* (COMP 642 — Machine Learning, Compuer Science, Rice University)

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In (p. 1-14). Computational and Biological Learning Society.

Sperduti, A., & Starita, A. (1997). Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, *8*(3), 714-735. doi: 10.1109/72.572108

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, *15*(56), 1929–1958.

Steinhaus, H. (1957). Sur la division des corps matériels en parties. *Bulletin de l'Académie Polonaise des Sciences*, *4*(12), 801–804.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... Rabinovich, A. (2015, June). Going deeper with convolutions . In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (p. 1-9). Los Alamitos, CA, USA: IEEE Computer Society.

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., & Mei, Q. (2015). Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web* (p. 1067–1077). Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee. Retrieved from https://doi.org/10.1145/2736277.2741093 doi: 10.1145/2736277.2741093

Thorndike, R. L. (1953). Who belongs in the family? *Psychometrika*, *18*(4), 267–276. Retrieved from https://doi.org/10.1007/BF02289263 doi: 10.1007/BF02289263

Tibshirani, R., Walther, G., & Hastie, T. (2002, 01). Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *63*(2), 411-423.

Tompson, J., Goroshin, R., Jain, A., LeCun, Y., & Bregler, C. (2015). Efficient object localization using convolutional networks. In *2015 ieee conference on computer vision and pattern recognition (cvpr)* (p. 648-656). doi: 10.1109/CVPR.2015.7298664

Tóth, L. (2013, 10). Phone recognition with deep sparse rectifier neural networks. In (p. 6985-6989).

Van den Ende, M., & Ampuero, J. (2020). Automated Seismic Source Characterization Using Deep Graph Neural Networks. *Geophysical Research Letters*, *47*.

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., & Solomon, J. M. (2019). Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics*, *38*(5), 146:1–146:12. doi: 10.1145/3326362

Weisfeiler, B., & Leman, A. A. (1968). The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series*, *2*(9), 12–16. (In Russian. English translation available in: *Alexander A. Razborov (Ed.), Papers in Computer Science*, 2012)

Wu, B., Liu, Z., Yuan, Z., Sun, G., & Wu, C. (2017). Reducing overfitting in deep convolutional neural networks using redundancy regularizer. In *International conference on artificial neural networks.*

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., ... Bengio, Y. (2015). Show, Attend and Tell: Neural image caption generation with visual attention. *International Conference on Machine Leaning*, 2048–2057.

Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2019). How powerful are graph neural networks? In *International conference on learning representations (iclr).*

Yang, C., Liu, Z., Zhao, D., Sun, M., & Chang, E. Y. (2015). Network representation learning with rich text information. In *Proceedings of the 24th international conference on artificial intelligence* (p. 2111–2117). AAAI Press.

Yano, K., Shiina, T., Kurata, S., Kato, A., Komaki, F., Sakai, S., & Hirata, N. (2021, May). Graph-partitioning based convolutional neural network for earthquake detection using a seismic array. *Journal of Geophysical Research: Solid Earth*, *126*.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., & Smola, A. J. (2017). Deep sets. In I. Guyon et al. (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc.

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2024). *Dive into Deep Learning.* Cambridge University Press.

Zhang, H., & Thurber, C. H. (2003, 10). Double-difference tomography: The method and its application to the hayward fault, california. *Bulletin of the Seismological Society of America*, *93*, 1875-1889.

Zhang, X., Reichard-Flynn, W., Zhang, M., Hirn, M., & Lin, Y. (2022). Spatiotemporal Graph Convolutional Networks for Earthquake Source Characterization. *Journal of Geophysical Research: Solid Earth*, *127*.

Zhang, X., Zhang, J., Yuan, C., Liu, S., Chen, Z., & Li, W. (2020). Locating induced earthquakes with a network of seismic stations in oklahoma via a deep learning method. *Scientific Reports*, *10*, 1941.

Zhang, X., Zhang, M., & Tian, X. (2021, 03). Real-Time Earthquake Early Warning With Deep Learning: Application to the 2016 M 6.0 Central Apennines, Italy Earthquake. *Geophysical Research Letters*, *48*.

Zhebel, O., & Eisner, L. (2015). Simultaneous microseismic event localization and source mechanism determination. *Geophysics*, *80*(6), KS1-KS9.

Zhou, D.-X. (2020). Universality of deep convolutional neural networks. *Applied and Computational Harmonic Analysis*, *48*(2), 787-794.

Zhu, W., & Beroza, G. C. (2018, October). Phasenet: a deep-neural-network-based seismic arrival-time picking method. *Geophysical Journal International*, *216*, 261-273.

Zhu, W., Mousavi, S. M., & Beroza, G. C. (2019). Seismic signal denoising and decomposition using deep neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, *57*, 9476-9488.

Zou, H., & Hastie, T. (2005, 03). Regularization and Variable Selection Via the Elastic Net. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *67*(2), 301-320.