Development of graph-based ML techniques for optimized particle analysis at LHC

A Thesis

submitted to
Indian Institute of Science Education and Research Pune
in partial fulfillment of the requirements for the
BS-MS Dual Degree Programme

by

Shreyas Bakare
Registration No. 20201192

Department of Physics,
Indian Institute of Science Education and Research Pune
Dr. Homi Bhabha Road,
Pashan, Pune 411008, INDIA.

March, 2025

Supervisor: Dr. Sourabh Dube

Certificate

This is to certify that this dissertation entitled **"Development of graph-based ML techniques for optimized particle analysis at LHC"** towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents study/work carried out by Shreyas Bakare at Indian Institute of Science Education and Research under the supervision of Dr. Sourabh Dube, Professor, Department of Physics, during the academic year **2024-2025**.

Dr. Sourabh Dube

Committee:

Dr. Sourabh Dube

Dr. Arka Banerjee

To my parents, for everything I am today is because of them.

Declaration

I hereby declare that the matter embodied in the report entitled Development of graph-based ML techniques for optimized particle analysis at LHC are the results of the work carried out by me at the Department of Physics, Indian Institute of Science Education and Research (IISER), Pune, under the supervision of Dr. Sourabh Dube and the same has not been submitted elsewhere for any other degree.

Shreyas Bakare

Acknowledgments

I am deeply grateful to my supervisor, Dr. Sourabh Dube, for his unwavering support, mentorship, and patience throughout this journey. Our discussions have shaped not only my approach to research but also my perspective on life. His guidance has been instrumental in building my confidence and motivation to pursue a PhD. I sincerely thank Dr. Arka Banerjee for his valuable insights as my TAC expert and the faculty of the Department of Physics at IISER Pune for their support and encouragement.

A special thanks to Arnab, Prachurjya, Yash, and Riya for their invaluable inputs and to Shriyansh, Kirutheeka, and Vaidehi for making long hours in the lab more enjoyable. I am also grateful to the entire EHEP group at IISER Pune. To all my friends, especially Shamant and Ritu, your constant support and reassurance have made this journey even more meaningful.

Above all, I am deeply thankful to my parents, Dr. Sunil Bakare and Dr. Shruti Bakare, and my sister, Dr. Shweta Bakare-Shendage, whose unwavering love, belief, and encouragement have been my greatest strength.

Finally, I acknowledge the Department of Science and Technology for the INSPIRE-SHE scholarship and IISER Pune for providing an enriching research environment.

Abstract

Machine Learning (ML) techniques have a long history of applications in High Energy Physics (HEP) experiments, such as at the Large Hadron Collider (LHC), which generates vast amounts of complex data. In this project, I explore the utility of graph-based ML methods, specifically Graph Neural Networks (GNNs) to develop novel data interpretation methodologies. GNNs leverage relational inductive bias by modelling data as graph structures. This offers a more expressive and adaptable data representation than conventional ML techniques that rely on image or sequence data. The key challenges of implementing a GNN for a particular task lie in constructing meaningful graph representations from LHC data, designing effective GNN architectures, and optimising these models for improved performance.

This thesis focuses on two applications of GNNs in HEP: (i) event classification and (ii) track-based analysis. In event classification, GNN-based framework is developed to distinguish signals from background events and the performance is compared against Deep Neural Networks (DNNs). And the impact of different graph structures, GNN operators, and architectural designs is investigated. Furthermore, I have developed a novel GNN algorithm that enhances classification performance by incorporating learnable edge weights.

In track-based analysis, I explore the potential of GNNs for processing low-level detector outputs. Particularly their ability to identify the presence of high transverse momentum charged particle from tracker hit information. Results demonstrate the advantages and challenges of GNN-based approaches in event classification and track-related tasks.

This research contributes to the growing adoption of geometric deep learning in particle physics and lays the foundation for future applications of GNNs in collider-based experiments.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The idea that all matter is composed of some indivisible units and the curiosity about these fundamental building blocks of nature dates back to ancient Greek philosophers Leucippus and Democritus. Over centuries, elementary particle studies have evolved drastically, with prominent breakthroughs shaping modern particle physics in the last hundreds of years.

Our current understandings of the fundamental laws of nature are encapsulated in a theoretical framework known as the standard model (SM) [1] of particle physics. SM classifies all known elementary matter particles into quarks and leptons and describes their interactions via the strong, weak, and electromagnetic forces. It accurately describes fundamental interactions across many orders of magnitude in energy. However, despite all its success and extensive experimental validation, SM remains incomplete. In fact, SM describes only about 4% of the universe's energy content, leaving the remaining 96%, composed of dark matter and dark energy, unexplained. It does not incorporate the fourth force of nature, gravity, nor does it explain questions such as neutrino masses and matter-antimatter asymmetry. A review of these open problems can be found in [2]. Addressing these gaps requires further theoretical and experimental advancements.

Experimental particle physics probes the fundamental particles and laws of nature by:

1. Producing and observing fundamental particles in controlled particle collisions.

2. Detecting astrophysical particles that naturally occur in the universe.

The study of astrophysical particles relies on passive observations, whereas controlled high-energy physics (HEP) collider experiments allow us direct and repeatable probes of various particle interactions.

Collider experiments, such as the CMS experiment at the Large Hadron Collider (LHC), focus on using statistical methods to test the fundamental laws of nature in controlled collisions of protons at nearly the speed of light. This involves billions of collisions per second. This enormous data is captured at the lower level as electronic signals from the detector output. These signals then undergo a higher-level physics object reconstruction process before being used in further physics analysis. Physics analysis involves understanding the underlying physics behind the observations. We utilize proton-proton collision simulations based on the relevant theoretical frameworks to compare the observations against simulations.

Given immense size and high dimensionality of the HEP data, advanced computational techniques have become essential for analysis. ML approaches are pivotal in improving tasks such as data filtration and collection, enhancing reconstruction and identification techniques, optimizing physics analysis for SM measurements and new physics searches involving signal-background event classification, ultimately helping physicists uncover new physics which may fill gaps in our understanding of nature at the most fundamental level.

HEP analysis commonly uses ML algorithms trained on relevant simulations, which are then applied to the data collected by the detector. Before deep learning, HEP analysis used ML techniques like boosted decision trees (BDT), support vector machine (SVM), etc. These methods helped boost many data analysis tasks. Still, in the literature [3], there is an opinion that they often failed to match the performance of physicist-engineered solutions, especially for high dimensional data. However, the recent advent of deep learning techniques has allowed ML tools to handle higher-dimensional, more complex problems than previously feasible.

Deep learning techniques like deep neural networks (DNN), convolutional neural networks (CNN) and recurrent neural networks (RNN) have proven valuable across many HEP analyses [4]. To use these deep learning techniques, it is necessary to first convert HEP data into vectors, images or sequences. This may not be the best representation of sparse HEP data [5].

Extending deep learning models to non-Euclidean input domains, generally called geo-

metric deep learning, is an emerging research area. Under this broad term, deep learning on graphs is becoming widely popular as graphs have more expressive power [6]. Graphs can naturally encode various types of particle physics data, such as energy deposits within a detector, individual physics objects like tracks or jets, individual particles or particle groups, and even heterogeneous information. While CNNs are well-suited for grid-structured data, such as image pixels, extend similar powerful techniques to irregular, graph-based data structures.

This thesis explores the applications of ML in HEP analysis, with a particular focus on graph-based ML techniques. I begin by introducing the core principles of GNNs, including the fundamental concept of graphs, the motivation for using graph-based ML techniques, and an overview of various GNN operators and architectural design choices. A systematic workflow for applying GNNs to HEP data is then outlined. This is followed by an in-depth investigation of two key tasks in HEP analysis:

- **Event classification:** Developing a GNN-based framework to distinguish signal events from the background, comparing its performance with DNN, and studying its dependence on different graph structures, GNN operators, and architectural designs. Additionally, this task examines the GNN properties, such as relational inductive bias and attention mechanisms, over traditional ML methods and identifies the challenges associated with their application.
- **Track-related analysis:** Exploring the potential of GNNs for analyzing low-level detector outputs, particularly their ability to handle variable input structures compared to conventional ML methods. Specifically, I investigate whether GNNs can identify the presence of high transverse momentum charged particles in an event by processing detector hit information.

Building upon insights from the attention mechanism, I have developed a GNN algorithm that incorporates learnable edge weights and leads to enhanced model performance in event classification compared to standard GNN architectures.

The thesis is organized as follows:

Chapter 2 provides an overview of the CMS experiment, detailing its detector components, data reconstruction techniques, particle four-momentum representations, and the

generation of event simulations. Chapter 3 discusses the role of ML in HEP, covering its applications, deep learning workflows, and model evaluation methods, concluding with an introduction to geometric deep learning. Chapter 4 introduces graph-based approaches, including GNN operators and architectural designs, and explains the fundamentals of GNNs in the context of HEP analysis. Chapter 5 presents the event classification task, focusing on $H \to ZZ$ versus ZZ classification, with a comparative study of basic GNN models and DNNs. Chapter 6 extends this analysis to different GNN architectures, assessing their impact on model performance and further exploring classifications such as $H \to WW$ versus WW, ZZ versus WW and $H \to ZZ$ versus $H \to WW$. Also delves into graph attention-based models and Section 6.5 introduces the novel GNN algorithm I developed to enhance classification performance. Chapter 7 examines the application of GNNs to track-based problems, particularly leveraging low-level detector outputs (tracker hits) to identify high transverse momentum charged particles. Finally, Chapter 8 summarizes the key findings, presents conclusions, and discusses future directions for GNNs in HEP analysis.

Chapter 2

Experimental setup

High-energy physics (HEP) collider experiments are typically conducted in big accelerator facilities like CERN, KEK, Fermilab, etc., where protons or other particles are accelerated to nearly the speed of light and collide at specific interaction points. These high-energy collisions produce all kinds of particles, including massive, rare and short-lived particles, whose properties must be carefully studied to extract new physics insights.

This thesis focuses on the CMS experiment at the LHC, CERN. This chapter briefly overviews LHC, the CMS experiment, and the simulations used to train the ML models in this thesis.

2.1 Large hadron collider

The LHC [7] at CERN is currently the largest and most powerful particle accelerator. It is located 100 meters beneath the Swiss-French border and spans 27 kilometres in circumference. Using superconducting magnets, the LHC accelerates two counter-rotating beams of proton bunches and collides them with energies of up to 13.6 TeV at four designated points along the circumference. Every bunch crossing results in multiple p-p collisions, with a bunch crossing frequency of 40 million times per second. These collision points host major HEP experiments like CMS [8], ATLAS [9], ALICE [10], and LHCb [11], each designed to capture and analyze different products of these high-energy interactions.

Since rare events, such as Higgs boson production, occur only once in several billion proton-proton collisions, LHC needs to operate at extreme rates, colliding nearly a billion protons per second. Most of the interesting particles produced at LHC decay before they can be directly observed. Therefore, analyses rely on reconstructing underlying interactions based on the properties of detectable decay products. This requires sophisticated, multi-layered detectors capable of capturing and processing vast amounts of data with high precision.

2.2 CMS detector

Figure 2.1 shows the CMS detector, a sophisticated, multipurpose particle detector at the LHC. It consists of multiple sub-detectors, each specifically identifying particles and measuring their properties. CMS sub-detectors primarily include a silicon tracker, electromagnetic calorimeter, and hadron calorimeter, all enclosed in a 3.8 Tesla superconducting solenoid magnet. Additionally, the CMS detector has multi-layered muon chambers placed outside the solenoid.

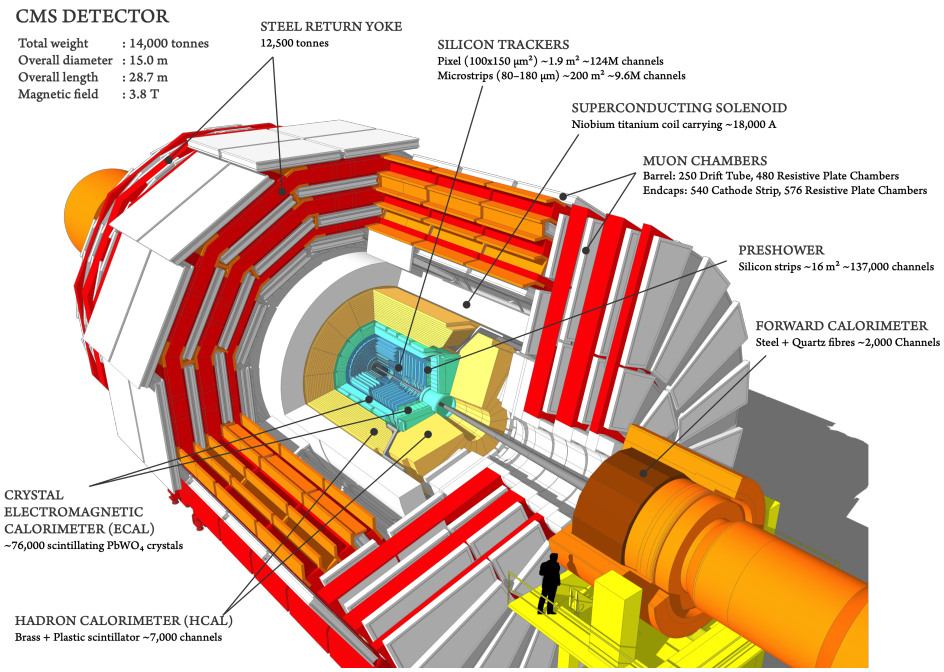


Figure 2.1: CMS detector at LHC [12]

1. **Silicon Tracker:** Silicon tracker is the innermost layer of the CMS detector. It is made up of silicon pixels and strips. When a charged particle passes through the silicon tracker, it ionizes the silicon atoms, giving us what we call hits in each tracker layer. Enabling the reconstruction of their trajectories and identification of their origin. This way, The tracker provides precise measurements of the momentum and charge of charged particles.

2. **Electromagnetic Calorimeter:** The electromagnetic calorimeter (ECAL) is a lead-tungstate crystal calorimeter designed to measure the energy deposits of electrons and photons.

3. **Hadron Calorimeter:** The hadron calorimeter (HCAL) is a brass and scintillator calorimeter designed to measure the energy deposits of hadrons.

4. **Muon Chambers:** The muon chambers are a set of drift tubes and cathode strip chambers designed to measure the trajectories of muons. This way, muon chambers provide precise measurements of the momentum and charge of muons.

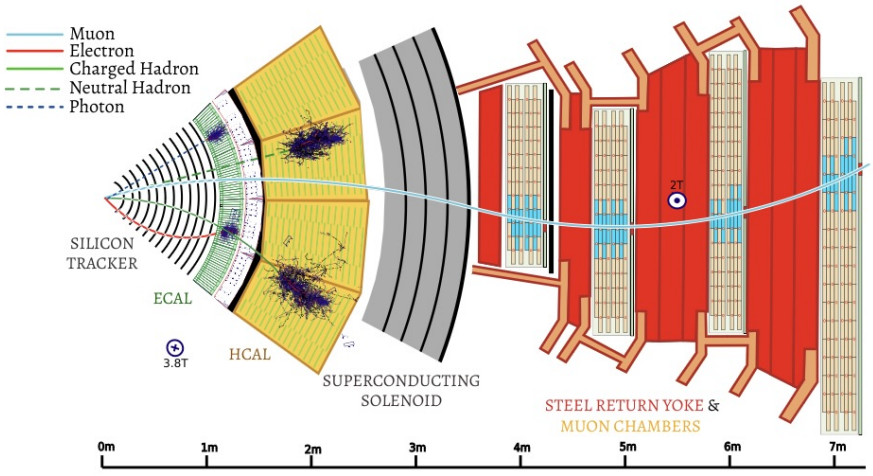


Figure 2.2: CMS sub-detectors and how different particles interact with them [8]

CMS sub-detectors and how each subdetector captures different particle properties are shown in Figure 2.2

The CMS detector has over 100 million detection elements, recording millions of high-dimensional measurements every second. The vast sensor arrays digitize these particle interactions, converting raw physical processes into an enormous volume of complex data. The measurements are inherently sparse in space and do not a-priori fit homogeneous, grid-like data structures. The challenge then lies in efficiently processing, filtering, and analyzing this data to extract meaningful insights.

2.3 Particle flow objects

We need to reconstruct physics objects from the low-level detector outputs. This is usually done with the particle flow (PF) [13] reconstruction algorithm. PF algorithm classifies all objects into five categories: electrons, photons, muons, neutral hadrons, and charged hadrons. Electrons are reconstructed by matching ECAL energy deposits to tracks from the silicon tracker. ECAL energy deposits with no matching track are classified as photons. Muons are reconstructed by matching tracks in the outer muon chambers with those in the inner silicon tracker. The momentum is calculated from the curvature of the track in the magnetic field. Energy deposits in the HCAL matching a track are classified as a charged hadron, while the one which does not find a matching track is classified as a neutral hadron. Jet clustering algorithms such as anti-kT [14] are then run over the PF objects to form PF Jets.

2.4 Particle representation

In HEP, the four-momentum of a particle is a fundamental quantity used to describe its energy and momentum components. Like other collider experiments, the CMS detector employs different representation systems to represent particles. Each representation has advantages depending on the type of analysis being performed.

Four momentum

The four-momentum P^μ of a particle is represented as:

$$P^\mu = (E, P_x, P_y, P_z), \tag{2.1}$$

where:

- E is the total energy of the particle,
- P_x, P_y, P_z are the momentum components along the three spatial axes.

This representation is more useful when dealing with exact kinematic calculations and theoretical formulations of relativistic dynamics.

Kinematic properties

In the CMS detector, a more natural representation for collider physics is the Kinematic properties, where the particle is expressed as:

$$P^\mu = (p_T, \eta, \phi, m), \tag{2.2}$$

where:

- $p_T = \sqrt{P_x^2 + P_y^2}$ is the transverse momentum, which is conserved in p-p collisions,
- $\eta = -\ln\tan\left(\frac{\theta}{2}\right)$ is the pseudorapidity, describing the angle of the particle relative to the beam axis,
- ϕ is the azimuthal angle measured in the transverse plane,
- m is the mass of the particle.

Since the CMS detector is a cylindrical detector centred around the beam axis, this representation is more practical for experimental analyses. In collider physics, pseudorapidity η is preferred over the polar angle θ because differences in rapidity are invariant under Lorentz boosts along the beam direction.

2.5 Event simulations

HEP event simulation plays a great role in understanding particle interactions and validating various theoretical models. Tools like MadGraph [15], Pythia [16], and Delphes [17] form a standard pipeline for simulating events from first principles to detector-level responses.

1. **MadGraph** generates hard-scattering matrix elements based on quantum field theory calculations, providing a foundation for simulating various physics processes. The output of MadGraph is an LHE file, which contains the initial state particles and their momenta. This is then passed to Pythia for further processing.

2. **Pythia** handles Parton showering, hadronization, and underlying event modelling. The output of Pythia is a HepMC file, which contains the final state particles and their momenta. This is then passed to Delphes for detector simulation.

3. **Delphes** performs fast detector simulations, incorporating resolution effects and reconstruction efficiencies to approximate the response of real experiments. The output of Delphes is a ROOT file, which contains the reconstructed PF objects like electrons, muons, jets, and missing transverse momentum.

Together, these tools enable precise modelling of high-energy collisions, facilitating comparisons between theoretical predictions and experimental data.

Chapter 3

Machine learning and HEP

High-energy physics (HEP) has a long history of ML applications [4] for various tasks, including data collection, physics object reconstruction and identification, SM measurements, new physics searches, etc.

Initially, HEP used ML techniques like boosted decision trees (BDT) [18], support vector machine (SVM) [19], etc. A very significant example of this is the use of BDTs in the LHC experiment that led to the Higgs discovery [20, 21]. Recent developments in deep learning architectures like Deep Neural Networks (DNN), Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNNs) have allowed more complex tasks to be performed on HEP data. The potential of deep ML approaches in HEP analysis can be found in [3].

In this section, I have introduced the basic framework of deep learning and discuss how to train, evaluate and assess the performance of deep learning models.

3.1 Deep learning pipeline

The complete deep-learning pipeline can be divided into three main optimization steps:

1. Input data

2. Learning / Training

3. Evaluation / Testing

3.1.1 Input data

These deep learning models operate on fixed-size inputs in Euclidean space. Different types of neural networks are designed for specific data structures. DNNs process fixed-size vector inputs. CNNs are designed for spatial data, such as images. And RNNs handle sequential or time-series data. The order of elements in input data is vital for all these models.

In the context of HEP, different deep-learning models are chosen based on the nature of the problem.

1. **Vector-based inputs (DNNs)**: Many HEP analyses represent events as fixed-size feature vectors containing various properties of particles. These features may include the number of jets and leptons, transverse momentum (p_T), pseudorapidity (η), azimuthal angle (ϕ), invariant mass (M), and missing transverse energy (E_T^{miss}) of particle systems. DNNs process such structured inputs effectively.

2. **Image-based inputs (CNNs)**: Raw detector outputs can be represented as images in certain tasks. For example, calorimeter energy deposits in jet reconstruction can be visualized as pixelated energy distributions. Similarly, tracking data from silicon detectors can be structured into 2D images, enabling CNNs to extract spatial correlations between hits and reconstruct jet substructures.

3. **Sequence-based inputs (RNNs/Transformers)**: Time-series or sequential data arise in scenarios where the temporal order of measurements is crucial. RNN models are well-suited for analyzing such sequential patterns.

3.1.2 Learning

Deep learning is a subset of machine learning that uses neural networks with many layers to learn data representations. The learning process has two broad classes:

1. Supervised ML: Learning from labelled data.
2. Unsupervised ML: To find patterns or structure in the data.

Supervised ML is concerned with predicting future outcomes from past experience. This means that each training example is paired with an output label. The goal of the model is to learn how to map inputs to outputs based on the labelled examples. This is in contrast to unsupervised learning, where the model is trained on data without labels and aims to find patterns or structures in the data. For this thesis, we focus on supervised ML. Supervised ML primarily performs two tasks:

1. Classification is the task of assigning a discrete label to an input.
2. Regression is predicting a continuous output value to an input.

Supervised learning has two main components:

1. Output score function: A function that takes the input features and produces a score for each class. This score is then used to make a prediction.
2. Loss function: A function that measures disagreement between prediction and the correct label.

Learning is minimizing the loss function, i.e., finding the best output score function that aligns with the correct label. This is achieved by adjusting the parameters of the output score function using backpropagation optimization algorithms such as gradient descent [22].

Neural networks

DNN is a type of neural network composed of multiple layers of neurons. Each neuron in a layer receives inputs from the previous layer, applies a weighted transformation, adds a bias,

and passes the result through a non-linear activation function σ. These layers collectively determine how the weights W and biases b are arranged to map the input to an optimal output. The deeper the network, the more complex hierarchical relationships it can learn. Enabling it to capture intricate patterns in the data. The final layer produces the output score function, which assigns a score to each possible class.

One simple example of a score function is a linear mapping:

$$f(x_i, W, b) = Wx_i + b \tag{3.1}$$

Where x_i represents the input features, and W and b are the model parameters, commonly referred to as weights and biases. In a classification task, the nature of x_i depends on the type of data: for a DNN, x_i could be a structured feature vector composed of physical properties (e.g., the number of jets, leptons, missing transverse momentum in HEP). For a CNN, x_i could represent an image where pixel intensities are flattened into a column vector before processing. The matrix W and vector b define how the input is transformed into output scores.

The output consists of scores assigned to each class, and a well-trained classifier is expected to assign the highest score to the correct category. The parameters W and b are learned during training using a labelled dataset. The learning process involves minimizing a loss function that quantifies how well the model's predictions match the true labels.

A commonly used loss function for binary classification is **Binary Cross-Entropy (BCE) Loss**, which is defined as:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right] \tag{3.2}$$

Where y_i is the true class label (0 or 1), $\hat{y}_i$ is the predicted probability, and N is the number of training samples. When the model is confident in a wrong prediction, the BCE loss penalizes incorrect classifications more, ensuring a smooth optimization process. BCE is a widely used loss function for deep neural networks in classification tasks.

Through backpropagation, the network continuously updates the weights and biases in each layer to reduce the loss. This process refines the decision, improving the model's ability

to generalize to unseen data. Only the learned parameters are required to predict new inputs once training is complete.

However, a single-layer linear model is often insufficient for capturing complex patterns in data. A DNN can learn more complex representations by stacking multiple layers of neurons. The initial layers may detect simple features, while deeper layers extract higher-level structures. Each layer refines the weights and biases based on the previous layer's outputs.

3.1.3 Evaluation

We train our model on a training dataset to learn the parameters W and b. We then evaluate this model (with pre-trained parameters) on a validation dataset to check its performance. The validation dataset is unseen that the model has not seen during training. This is to check if the model is overfitting or underfitting. Overfitting happens when a model is too complex and learns the training data too well, including noise, making it struggle with new data. Underfitting, on the other hand, occurs when a model is too simple and fails to capture important patterns, leading to poor performance overall. This way, we can further tune and retrain our model for optimal performance.

3.1.4 HEP Analysis

HEP experiments often use supervised ML to learn complicated inverse functions. The goal is to extract insights about the underlying physics process using the data collected from the detector. This idea is illustrated in Figure 3.1.

First, we generate a simulation of the physics process we are interested in. This simulation is then passed through a detector simulation to generate a detector response. Here, the simulation acts as a "truth record" of the physics event. This "truth record" is used to train supervised learning algorithms to reverse the detector simulation and uncover the underlying physics from the observed data. Once trained, these algorithms are applied to real detector measurements.

Figure 3.2 illustrates a classification task using a DNN model (More about this classifi-

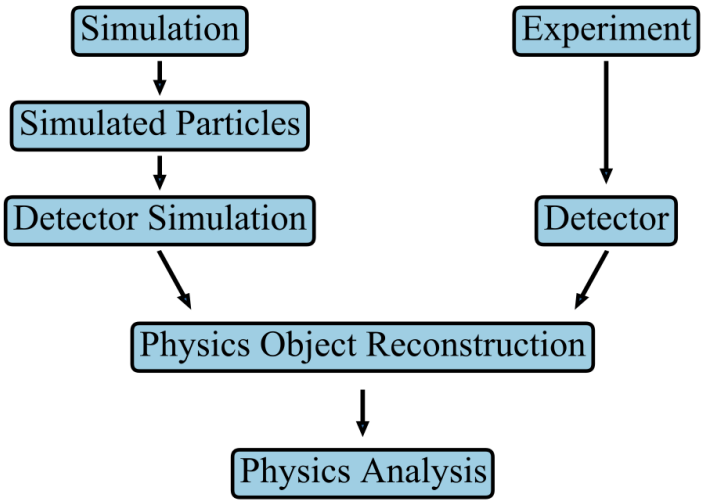


Figure 3.1: HEP analysis workflow [5]

cation problem in Chapter 5). Here, the histogram of the Ouput scores shows signal peaks at around 1, whereas background peaks at around 0. Giving us a good separation. We need to cut this score to get a good signal-to-background ratio.

3.1.5 Receiver Operating Characteristic (ROC) curve

The performance of a classifier depends on where you cut the score, known as the threshold. By varying this textbfdecision threshold, we can achieve different textbfsignal-to-background ratios. This variation is captured in a Receiver Operating Characteristic (ROC) curve.

The ROC curve is a plot of the True Positive Rate (TPR), also known as **signal efficiency**, against the False Positive Rate (FPR), or **background efficiency**, at different threshold values. These metrics are defined as:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.3}$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{3.4}$$

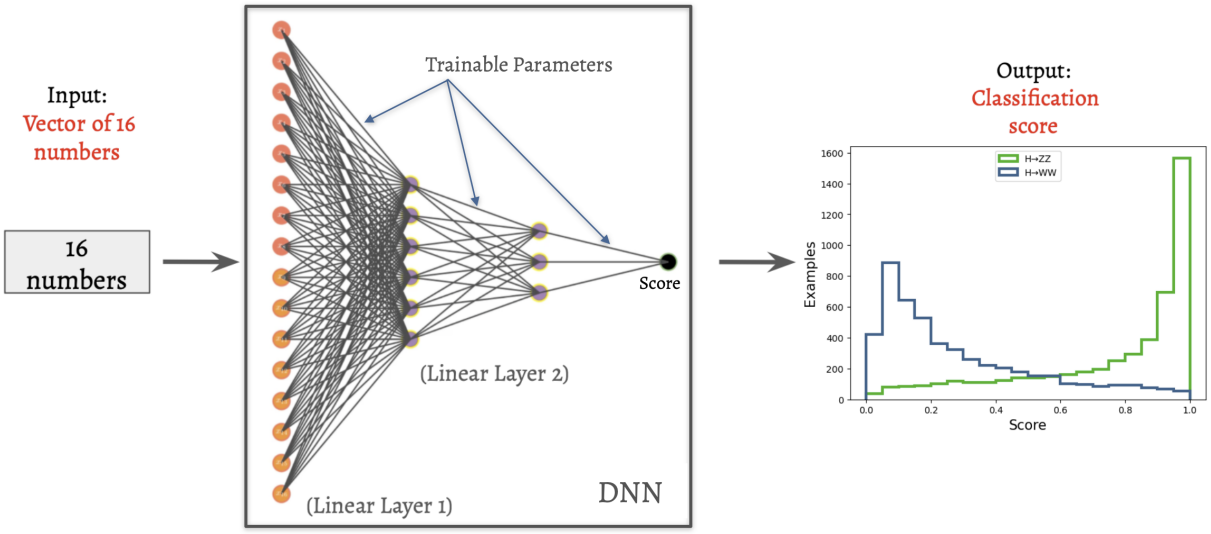


Figure 3.2: Example of a DNN for classification task

Where:

1. TP(True Positives): Correctly identified signal events.
2. FN (False Negatives): Signal events misclassified as background
3. FP (False Positives): Background events misclassified as a signal
4. TN (True Negatives): Correctly identified background events

The ROC curve illustrates the trade-off between TPR and FPR. A classifier with perfect performance would have a curve reaching the top-left corner (TPR $= 1, \text{FPR} = 0$), indicating high signal efficiency with minimal background contamination.

To summarize the classifier's performance in a single metric, we compute the area Under the Curve (AUC) A higher AUC value (AUC ≈ 1) signifies a strong classifier, while an AUC close to 0.5 indicates a model performing no better than random guessing.

Figure 3.3 illustrates how ROC represents different cuts on the DNN score. Here, the red cut is a strong cut that gives a purer signal, i.e., low background efficiency, but as a result, we get low signal efficiency, i.e., we miss a lot of signal events. While the purple cut is a loose cut, it gives high signal efficiency, but, as a result, we get high background efficiency,

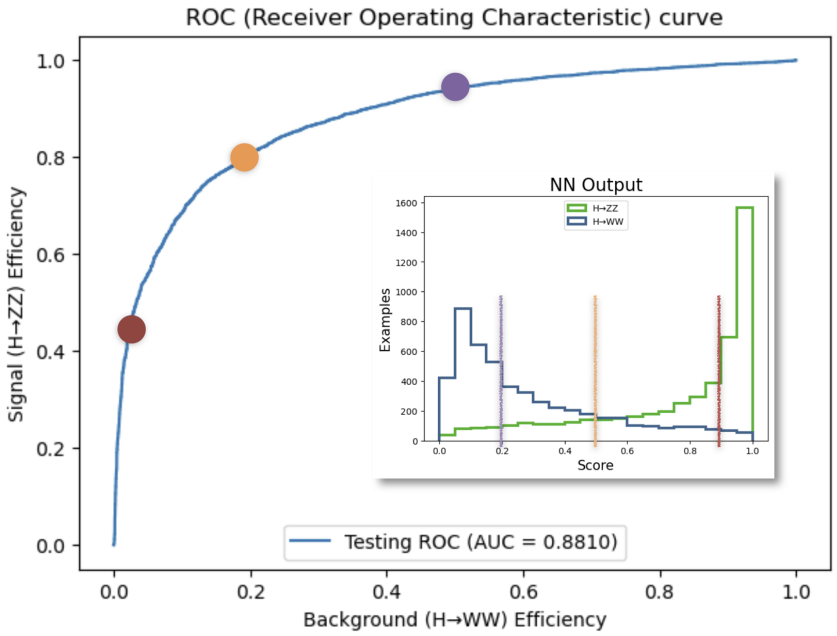


Figure 3.3: Illustration explaining ROC for the DNN model described in figure 3.2

i.e. we get a lot of background events mimicking signal events. And so on. The AUC for this model is 0.88, which is essential to compare performance with other models (More about this classification problem in Chapter 5).

3.2 Geometric deep learning

Geometric Deep Learning (GDL) [23] extends deep learning beyond traditional Euclidean domains by leveraging fundamental symmetry principles such as translations, permutations, and isometries. GDL categorizes data into five geometric domains: Grids, Groups, Graphs, Gauge and Geodesics, commonly referred to as the 5 Gs of geometric domains. Table 3.1 summarizes different domains and their associated symmetries. This framework provides a unified approach to generalizing deep learning across various structured and unstructured data types.

In essence, the geometric structure of the input domain, along with known symmetry groups, provides three fundamental building blocks for learning:

Domain	Example Data Structure	Symmetry Operator
Grids	Image data	Translations (Shift-invariance)
Groups	Homogeneous spaces	Group transformations
Graphs	Social networks, molecules	Permutations (Permutation-invariance)
Gauge and Geodesics	3D meshes, manifolds data	Isometries (Rotations, translations)

Table 3.1: GDL domains and their corresponding symmetry operators [23].

1. Local equivariant maps - Ensure that transformations preserve relationships.
2. Global invariant maps - Extract higher-level representations that remain consistent under symmetry transformations.
3. Pooling operators - Aggregate local features into global descriptors.

Graphs-based neural networks are the most general class of GDL that operate on graph-structured data. They are designed to capture the complex relationships and dependencies between input data, making them well-suited for tasks such as:

1. Graph classification
2. Node classification
3. Link prediction

For this thesis, we focus on graph classification tasks. Figure 3.4 is an illustration of the GDL blueprint. It shows the steps involved in getting a graph-level representation from a graph. This representation is used for graph classification tasks.

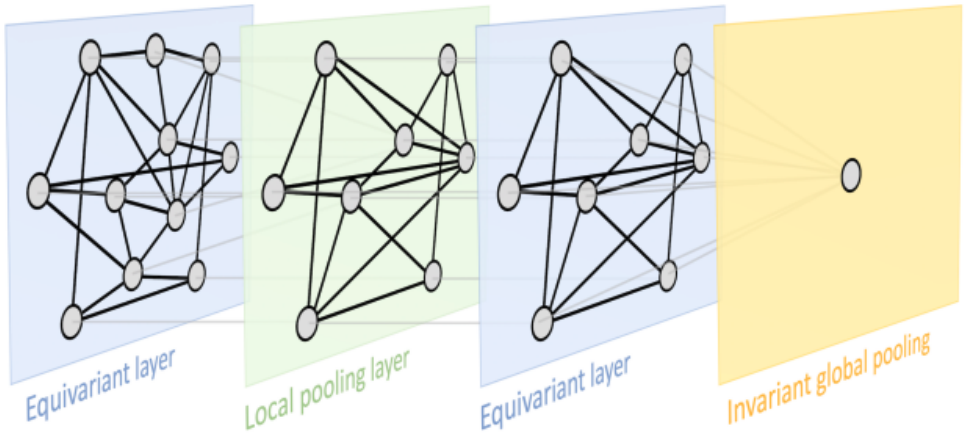


Figure 3.4: GDL blueprint, illustrating a graph-level representation [23]

Chapter 4

Graph Neural Network

Graph Neural Network (GNN) is a class of GDL that extends deep learning to relational data represented as graphs, where relationships between entities are as important as the entities themselves. Unlike traditional neural networks, which operate on grid-like data (e.g. vectors for DNNs, images for CNNs or sequences for RNNs), GNNs are naturally more adaptable to various problems in high-energy physics (HEP) analysis.

One of the most powerful aspects of GNNs is their flexibility. Depending on the graph structure, they can be reduced to simpler architectures like point sets, CNNs, RNNs and transformers [24].

In this chapter, I have introduced the fundamental concepts of graphs, the core workflow of GNNs, key GNN operators, and post-processing techniques like pooling. This provides a strong foundation for understanding their application in HEP and beyond.

4.1 What are graphs?

Graph is a data structure that describes a set of objects or elements known as graph nodes and their pairwise relationships/ connections known as graph edges. Graph nodes have node features, and graph edges have edge attributes. Along with these two, a graph has a label for the entire graph, which is known as a universal graph attribute. These graph labels are used

for graph classification tasks. An example of a graph with four nodes, each with four-node features and five edges, is shown in Figure 4.1.

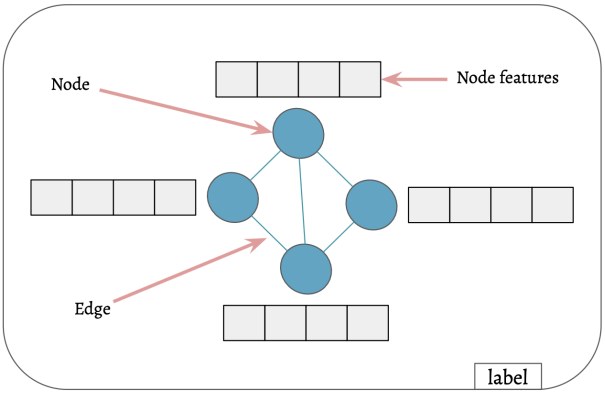


Figure 4.1: Example of a graph with four nodes and five edges.

1. A graph G is defined as $G = (u, X, E)$, where u is the universal graph attribute, X is the set of node features, and E is the set of edge attributes. Here, graph G has N_x nodes and N_e edges.

2. All the node-related information is represented with the node feature matrix $X = \{x_i\}_{i=1:N_x}$, where x_i is the feature vector of node i. The feature vector x_i is a f dimensional vector, i.e. every node has f features.

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1f} \\ x_{21} & x_{22} & \cdots & x_{2f} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N_x1} & x_{N_x2} & \cdots & x_{N_xf} \end{bmatrix} \tag{4.1}$$

   Node feature matrix is of order $N_x \times f$.

3. All the edge-related information is represented with the adjacency matrix $E = \{e_{ij}\}_{i,j=1:N_x}$, where e_{ij} is the edge attribute vector of the edge between node i and node j. The edge attribute vector e_{ij} is a a dimensional vector. i.e. every edge has a attribute.

4. To keep it simple, we are considering $a = 1$, i.e. every edge has only one attribute. We call it weight. Weight $= 0$ means the edge is not present. The higher the weight, the

stronger the edge.

$$E = \begin{bmatrix} e_{11} & e_{12} & \cdots & e_{1N_x} \\ e_{21} & e_{22} & \cdots & e_{2N_x} \\ \vdots & \vdots & \ddots & \vdots \\ e_{N_x1} & e_{N_x2} & \cdots & e_{N_xN_x} \end{bmatrix} \tag{4.2}$$

Adjacency matrix is a square matrix of order $N_x \times N_x$.

5. Therefore, a graph G has a label u along with a node feature matrix X of order $(N_x \times f)$ and an adjacency matrix E of order $(N_x \times N_x)$.

4.2 Why graphs?

Graphs are all around us. They are a more natural way to represent data. Here are a few reasons why graphs are useful for HEP data representation.

1. Many problems in HEP involve data as unordered sets of elements with some relations in between. This data can be expressed more naturally as graphs, representing data points as nodes. These nodes are connected by edges, capturing the interactions between them.

2. GNNs have permutation invariance, which means that the model's predictions remain unchanged if the order of nodes or edges is altered. This is important, as graph data lacks a natural ordering like images or sequences.

3. In particle physics, graph-based representations provide several advantages over traditional data formats. Unlike vector or grid-based structures, graphs allow variable-sized data without the need for information loss.

4. Additionally, graphs are well-suited for handling sparse and heterogeneous detector data that may be difficult to represent using image-based methods and do not require artificial ordering like sequence-based representations.

5. As shown in Figure 4.2, graphs can can be reduced to other data structures. A graph without any edge is referred to as a point cloud. A graph with edges connecting nodes linearly forms a sequence. GNN on such a graph is analogous to RNN. A graph

with edges creating a grid-like structure represents an image; GNN on such a graph is analogous to CNN. Lastly, a graph where each node is connected to every other node by an edge is analogous to a transformer architecture. Making GNNs a more general deep-learning framework.

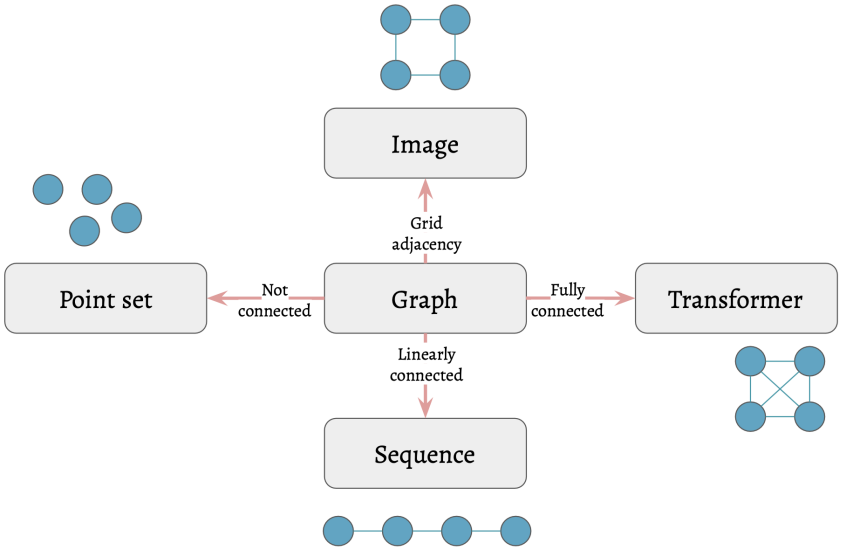


Figure 4.2: Reduction properties of a graph

4.3 Pipeline for GNN

GNN learns by aggregating information across each node's local neighbourhood, which is defined by its incident edges. GNNs produce node-level, edge-level, or graph-level predictions; these predictions are leveraged by a subsequent post-processing algorithm for node classification, link prediction, or graph classification tasks, respectively.

Workflow to build a GNN architecture for an HEP problem is a three-step process:

1. **Graph construction**: Making appropriate graphs from HEP data.
2. **Optimising graph networks (GN)**: Learning the graph-to-graph functions.
3. **Post-processing**: Making predictions from the output graphs of GN.

4.4 Graph construction

For a particular problem in HEP, there are many ways to construct graphs from the data. First, we need to identify nodes from the HEP data. Once we have the nodes, we can make various graphs by connecting the nodes with edges. Unlike other fields, HEP does not have a graph dataset library. To address this, I have developed a Python-based grapher class to help generate custom-made PyTorch Geometric graph datasets [25]. Grapher takes in the HEP data as a CSV file along with the graph construction parameters and returns a graph dataset. Graph construction is a crucial step, and we objectively study the effect of input graph structure on the model performance in the upcoming chapters.

4.5 Graph networks

How DNN transforms the input vector into hierarchical representations for classification, GNN applies similar principles to graph-structured data. Instead of processing independent numbers of vector input, GNNs operate on nodes and edges, propagating information across the graph to learn meaningful representations. The GNN flow diagram is shown in Figure 4.3.

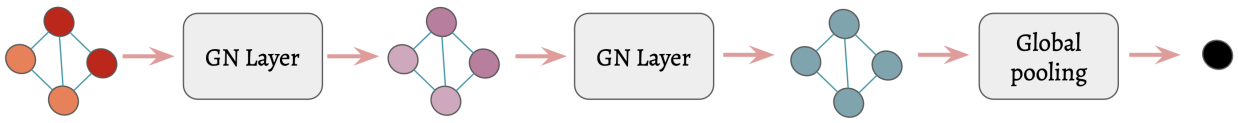


Figure 4.3: Flow diagram for the GNN architecture using two GN layers.

4.5.1 Types of GN operators

GN operators are graph-to-graph functions that transform an input graph into an output graph. These operators play a crucial role in how information is propagated and aggregated across the graph structure. Broadly, GN operators can be classified into three flavours:

As illustrated in Figure 4.4, the three main types of GNN operators differ in how they

aggregate information from neighbouring nodes, from left to right:

1. **Convolutional network**: These extend the concept of CNNs to graphs by aggregating features from local neighbourhoods. The sender node features are multiplied by a constant weight, c_{uv}.

2. **Attentional network**: The aggregation weight is dynamically computed via an attention mechanism of the receiver node over the sender node, $\alpha_{uv} = a(x_u, x_v)$.

3. **Message-passing network**: Vector-based messages are computed as a function of both the sender and receiver node features, $m_{uv} = \psi(x_u, x_v)$.

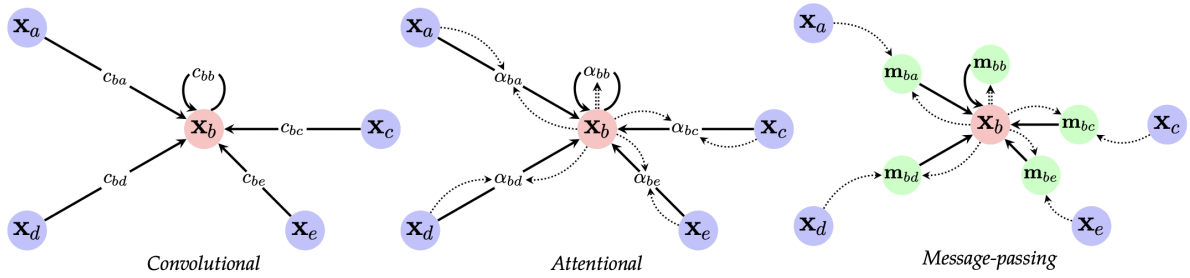


Figure 4.4: Visualization of the dataflow for the three flavours of GNN [23]

Convolutional network

Graph Convolutional Network (GCN) generalizes traditional convolution to graphs by leveraging spatial methods to aggregate neighbouring node information.

GCNConv layer One of the most widely used convolutional layers in GNNs is the GCNConv layer based on the paper by Kipf et al. [26], which updates node features using the normalized adjacency matrix:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{4.3}$$

where:

1. $H^{(l)}$ and $H^{(l+1)}$ are node embeddings at layers l and $l+1$,

2. $\tilde{A} = A + I$ is the adjacency matrix with self-loops,

3. $\tilde{D}$ is the degree matrix of $\tilde{A}$,

4. $W^{(l)}$ are learnable weight parameters, and

5. σ is the activation function (e.g., ReLU).

GraphConv layer Unlike GCNConv, the GraphConv layer, based on the paper by Morris et al. [27], explicitly learns edge-wise transformations:

$$H^{(l+1)} = \sigma\left(W^{(l)}H^{(l)} + \sum_{j\in\mathcal{N}(i)} W_e H_j^{(l)}\right) \tag{4.4}$$

where W_e captures edge-specific transformations and $\mathcal{N}(i)$ denotes the neighbourhood of node i.

Attentional network

Instead of uniform aggregation, Graph Attention Network (GAT) dynamically weights neighbouring nodes based on learned attention coefficients. This enables the model to focus on more relevant information.

GATConv layer The GATConv layer, based on the paper by Veličković et al. [28], applies an attention mechanism to determine the importance of each neighbour. i.e. The model decides which nodes are more important to focus on. The attention coefficients are computed as follows:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(a^T[Wh_i \,||\, Wh_j]\right)\right)}{\sum_{k\in\mathcal{N}(i)} \exp\left(\text{LeakyReLU}\left(a^T[Wh_i \,||\, Wh_k]\right)\right)} \tag{4.5}$$

$$h_i' = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} W h_j\right) \tag{4.6}$$

where:

1. α_{ij} represents the attention coefficient between nodes i and j,
2. W is the learnable transformation matrix,
3. a is the attention vector,
4. $||$ denotes concatenation, and
5. σ is an activation function.

4.6 Post processing

In graph classification tasks, a GNN first transforms the input graph into a latent graph representation using these graph network operators. So, the input graph is now transformed into a latent graph representation. We now need to perform the post-processing steps to get the final output. This involves aggregating the node-level features into a global graph representation and then applying a classification to predict labels. The output classification performance is then analyzed like traditional DNNs, using ROC curves. We only look at the global post-processing steps, such as Max [29], Mean [30], and Add [31] pooling.

4.7 GNN and HEP

The choice of GNN architecture determines how the graph data is processed and how the features are extracted.

Here, in Table 4.1, I list some interesting HEP applications of GNNs

Task Type	**Examples**
Event-level tasks	Event classification Anomaly detection Vertex reconstruction Improving triggers Regressing missing particles
Object-level tasks	Track reconstruction Jet clustering & identification Particle identification Energy reconstruction Particle flow reconstruction

Table 4.1: Some interesting HEP problems for GNNs

Chapter 5

Event classification task

Event classification is a key task in high-energy physics (HEP) experiments, playing a crucial role in identifying events with rare processes while suppressing abundant background events. In many physics analyses, the signal process of interest (P_1) and a background process (P_2) yield similar signatures, i.e. produce similar final-state particles, making them difficult to distinguish. A typical challenge is when the background occurs at a much higher rate than the signal. Say, P_2 is produced ten times more frequently than P_1. Event classification aims to maximize the identification of P_1 events while rejecting as many P_2 events as possible.

This classification task directly impacts the sensitivity of a search experiment, often quantified by the signal-to-background ratio:

$$\frac{N_S}{\sqrt{N_B}} \tag{5.1}$$

where, N_S is the number of (correctly) identified signal events, and N_B represents the background contamination. Maximizing this ratio strengthens our ability to detect potential new physics.

Traditionally, HEP uses a cut-flow-based analysis technique, where physicists manually define selection criteria based on kinematic and detector-level observables. However, HEP experiments like ATLAS and CMS have used ML-based analysis techniques for the past few decades to classify signal events from background events [32]. These methods optimize

classification performance, leading to improved background rejection. The question we now explore is, can we use GNNs for this task? If we can, what are the advantages and challenges? This method allows us to leverage relational information, potentially enhancing classification performance.

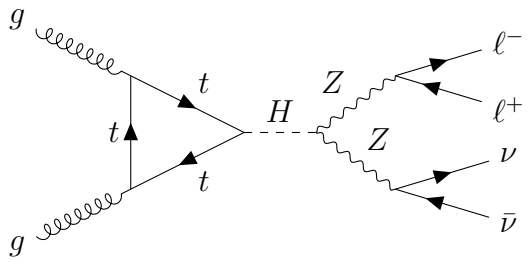
This chapter outlines the overall workflow to develop a baseline event classification model using GNN, starting from generating proton-proton collision simulations for signal and background events. Each event is then represented as a graph, with signal and background events forming two different graph classes. A simple GNN model is then trained to classify these events as a graph classification problem, outputting a probability score where 1 represents a signal event, and 0 represents a background event. We compare the performance of our GNN model with a DNN and objectively assess the advantages and challenges of using GNNs in HEP analysis. In the next chapter, I have explained an extensive comparison of various design choices and their impact on classification performance.

This exercise gives us a good understanding of how different design choices influence the performance of GNN models, making us aware of how to choose the right GNN architecture for a given task. It has equipped us well for future HEP analysis involving GNNs.

5.1 Generating event simulations

In the first step, we generate proton-proton collision simulations for signal and background events. As explained in Chapter 2, We used the MadGraph5 framework to generate these event simulations. Then, we used the Pythia8 for hadronization and Delphes for CMS detector simulation. We generated 10000 events for each process. We used various SM processes like ZZ, WW, $H \to ZZ$, $H \to WW$, WZ and $t\bar{t}$ processes.

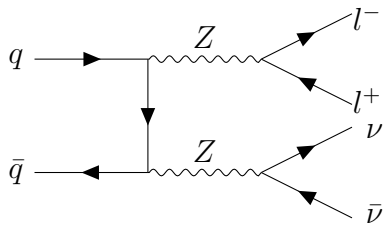
Lets begin with $H \to ZZ$ versus ZZ classification, where signal process is:



(5.2)

$$H \to ZZ \to \ell^-\ell^+\nu\bar{\nu}$$

In this process, p-p collision produces a Higgs boson (H), which then decays into two Z bosons. One of the Z bosons further decays into a lepton-antilepton pair ($\ell^-\ell^+$, where ℓ represents either an electron or a muon), while the other Z boson decays into a neutrino-antineutrino pair ($\nu\bar{\nu}$). The background process is:



(5.3)

$$ZZ \to \ell^-\ell^+\nu\bar{\nu}$$

Here, the p-p collision produces two Z bosons. One of which further decays into a lepton-antilepton pair ($\ell^-\ell^+$, where ℓ represents either an electron or a muon), while the other Z decays into a neutrino-antineutrino pair ($\nu\bar{\nu}$) leading to the same final state as the signal: $\ell^-\ell^+\nu\bar{\nu}$.

Each class contains 10,000 events, resulting in 20,000 events overall. These events are evenly split into a **training** and **testing** dataset, with 5,000 signal and 5,000 background events in each subset.

5.2 Graph representation of HEP events

Each event is represented as a graph, where we treat each of these four final-state particles as nodes in the input graph. Their four-momentum components serve as node features. These features can be either four momentum of the particles (p_x, p_y, p_z, E) or kinematic properties (p_T, η, ϕ, M). For now, we choose the four momentum. Comparisons between both are shown in Chapter 6. Figure 5.1 illustrates the distribution of node features p_x, p_y, p_z, and E for all four final-state particles, sorted by transverse momentum (p_T), for both $H \to ZZ$

(one) and ZZ (zero) events. In Figure 5.1, histograms in each row correspond to the four-momentum components (p_x, p_y, p_z, and E, from left to right) for the different particles in the final state: the leading lepton, subleading lepton, leading neutrino, and subleading neutrino (from top to bottom).

To ensure stable training, node features are normalized using min-max scaling with predefined ranges. The maximum and minimum values for each feature are provided in Table 5.1.

Table 5.1: Normalization ranges for node features p_x, p_y, p_z, and E

Particle	**Feature**	**Range (Min, Max)**
Leading Lepton	p_x	(-200, 200)
	p_y	(-200, 200)
	p_z	(-1000, 1000)
	E	(0, 1000)
Subleading Lepton	p_x	(-75, 75)
	p_y	(-75, 75)
	p_z	(-750, 750)
	E	(0, 750)
Leading Lepton Neutrino	p_x	(-200, 200)
	p_y	(-200, 200)
	p_z	(-1000, 1000)
	E	(0, 1000)
Subleading Lepton Neutrino	p_x	(-75, 75)
	p_y	(-75, 75)
	p_z	(-750, 750)
	E	(0, 750)

We have four nodes in the graph for our event classification task from four leptons in the final state. Even after defining our nodes and their features, multiple input graph structures can be constructed by connecting nodes differently. For example, we can connect all the nodes to each other, or we can connect only some of them. We can also connect the nodes in a way that reflects the physical properties of the event, etc. The choice of graph structure plays an important role in model performance, as graph structure dictates how information propagates between nodes. A detailed comparison of the impact of various graph structures is presented in Chapter 6. Let's choose the simplest possible configuration, a graph with no edges, treating the final-state particles as an unconnected point cloud.

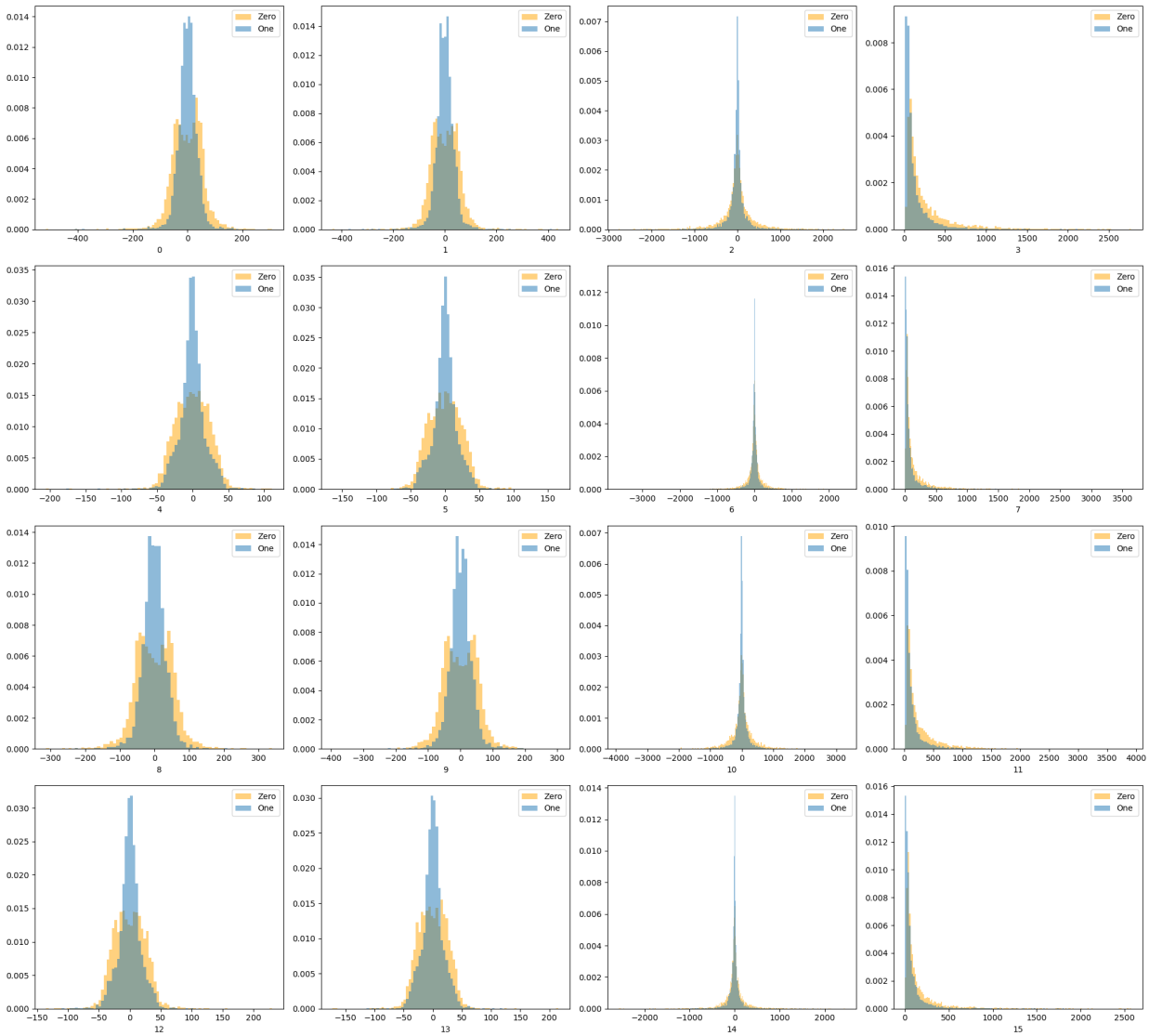


Figure 5.1: Horizontal axis: p_x, p_y, p_z, E (4-vectors) for; vertical axis: all four particles sorted by p_T for both $H \to ZZ$ (one) and ZZ (zero) events

5.3 GNN architecture

As discussed in Chapter 4, the GNN architecture consists of GN operators and a pooling layer, each with multiple possible choices. We use the GCNConv layer with global max pooling for our initial model. A comprehensive comparison of different graph convolution operators and pooling methods is shown in Sections 6.1.3 and 6.1.2, respectively.

5.4 Model optimisation

Now that the network design is fixed, the next step is to optimize the model for better performance. This involves selecting the correct number of trainable parameters, including the number of GNN layers and the latent dimensionality of the graph embeddings. Additionally, we fine-tune key hyperparameters such as batch size, number of epochs, and learning rate to achieve optimal results. For optimization, we use the Adam optimizer [33] and the binary cross-entropy loss function (explained in Chapter 3), which are well-suited for our binary classification task.

The only way to optimize the model is through systematic experimentation. We explore different numbers of GNN layers and hidden dimensions across various sets of hyperparameters, evaluating their impact on performance. By iterating through these configurations, we identify the most effective combination that yields the best results.

As shown in Figure 5.2, increasing the number of epochs leads to only a marginal improvement in AUC (around 2%). However, the training time scales linearly, increasing from 3 minutes for 50 epochs to approximately 90 minutes for 1500 epochs. Hence, choosing 50 epochs provides a reasonable trade-off between performance and time. Similarly, as illustrated in Figure 5.3, AUC tends to decrease with increasing batch size, while training time decreases exponentially. Considering this, we select a batch size of 40 as the optimal choice. A similar approach is followed to tune other parameters.

We finalize the hyperparameters for further comparisons, as summarized in Table 5.2. It is important to note that each data point in these plots represents a separately trained model with the corresponding hyperparameter configuration.

Table 5.2: Final choice of hyperparameters

Hyperparameter	**Chosen value**
Number of epochs	50
Batch size	40
Learning rate	0.005
Model parameters	(Case specific)

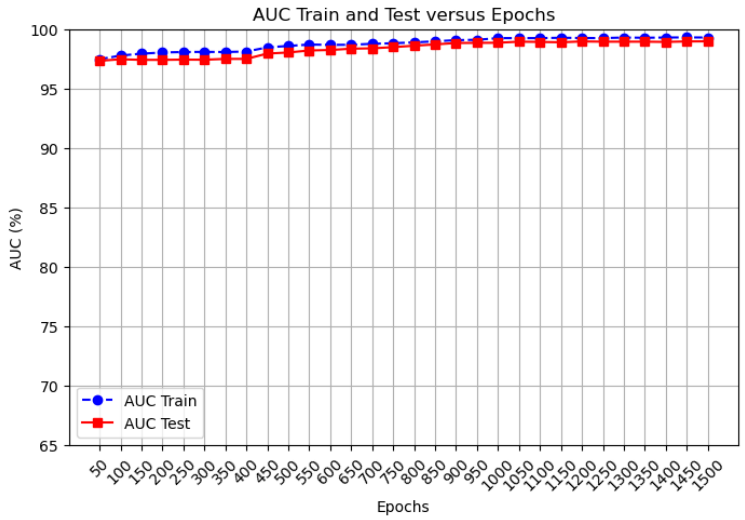


Figure 5.2: AUC vs Number of epochs

5.5 Model performance and ROC

Having optimized our hyperparameters, we now assess the performance of our trained GNN in classifying H $\rightarrow$ ZZ (signal) and ZZ (background) events. Let's summarize our key steps so far:

1. Defined the event classification problem in HEP as a graph classification task.

2. Constructed graphs with four nodes (leading/subleading leptons and neutrinos) using four-momentum components as node features with no edges in between (point-set).

3. Designed a baseline GNN model with GCNConv layers and global max pooling.

4. Tuned design parameters like the number of layers, hidden dimensions, epochs, batch size, etc.

Now, we visualize the model's output and classification performance.

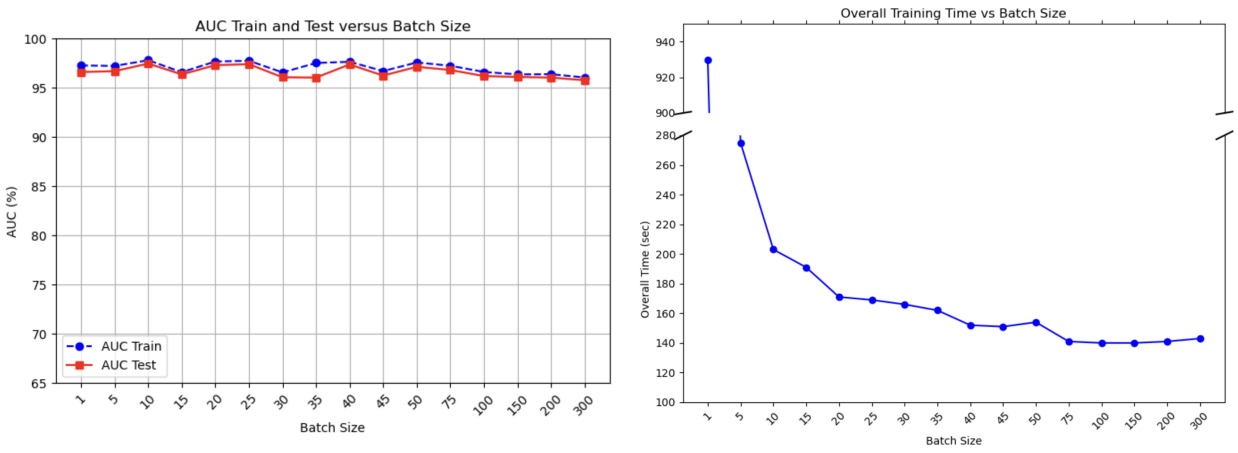


Figure 5.3: (a) AUC vs Batch size and (b) Training time vs Batch size

5.5.1 GNN output score distribution

Figure 5.4 shows the distribution of output scores for signal and background events. The solid histograms represent the test dataset, while the dashed histograms correspond to the training dataset. The x-axis represents the model's output score, where 1 corresponds to signal-like events and 0 to background-like events. It's also worth noting that the training and testing output scores have similar shapes, suggesting consistent model behaviour across both seen and unseen datasets.

5.5.2 ROC curve

The ROC curve in Figure 5.5 summarizes the classification performance. The blue curve represents the ROC for the training dataset, while the orange curve corresponds to the ROC for the test dataset. The AUC scores for the training and test datasets are 96.5% and 96%, respectively, indicating strong classification performance. The close agreement between the two suggests that the model is not overfitting and performs consistently across unseen data. From now on, I'll include only the test ROC curves in the thesis.

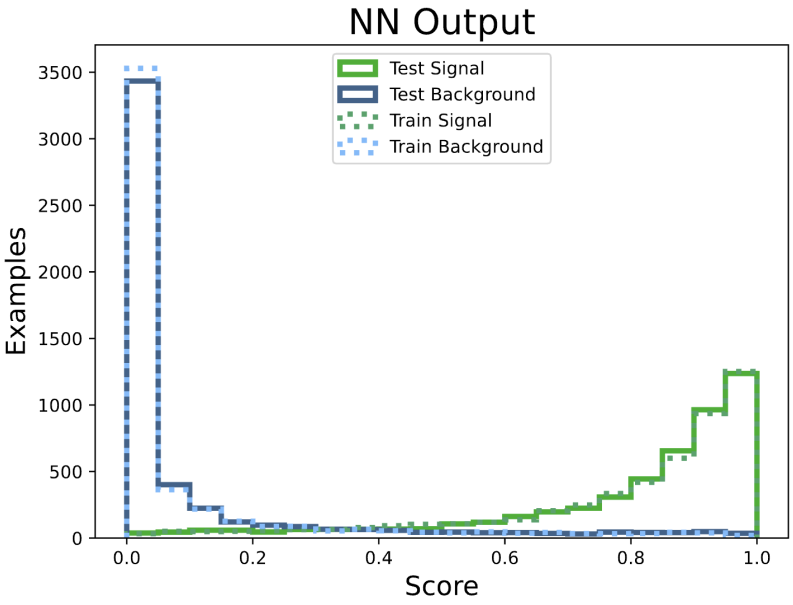


Figure 5.4: GNN output for H → ZZ versus ZZ

5.6 Comparison with DNN

To benchmark the performance of our GNN-based approach, we compare it against a DNN with a similar number of trainable parameters and two linear hidden layers. Unlike the GNN, which operates on graph-structured data, the DNN takes a vector of 16 event-level features as input. Figure 5.6 presents an overlay of the ROC curves for the test dataset for both models, demonstrating that even our simplest GNN model outperforms the DNN for a comparable parameter count. In the next chapter, we systematically study various design choices, such as graph structures, pooling methods, convolutional layers, and coordinate representations, to enhance classification performance further.

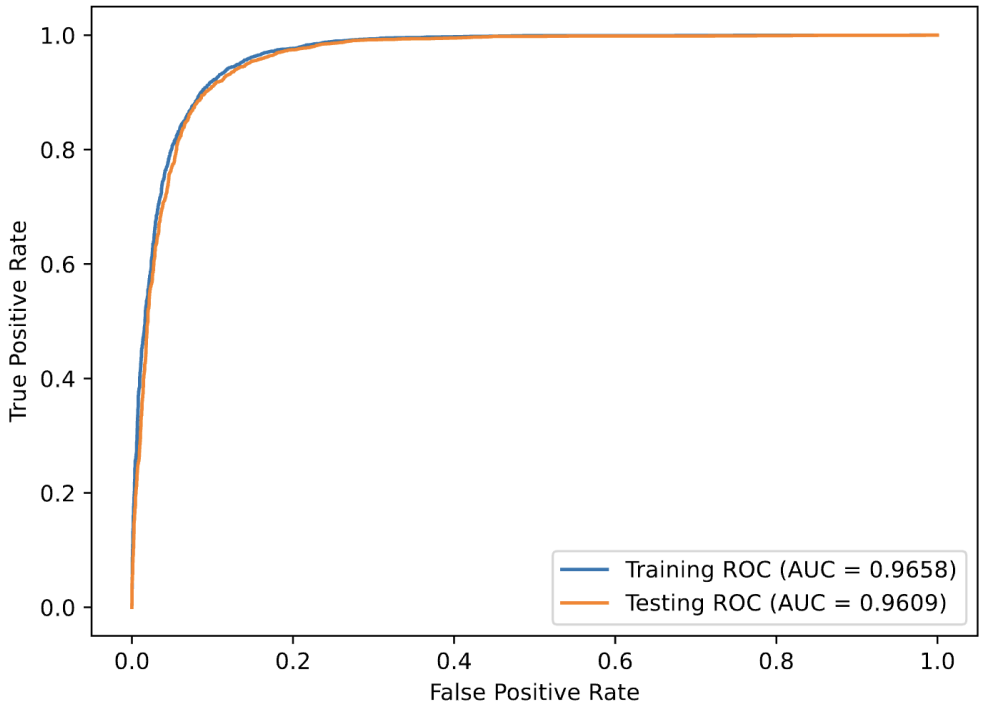


Figure 5.5: Train and test ROC for H $\rightarrow$ ZZ versus ZZ using GNN on point cloud

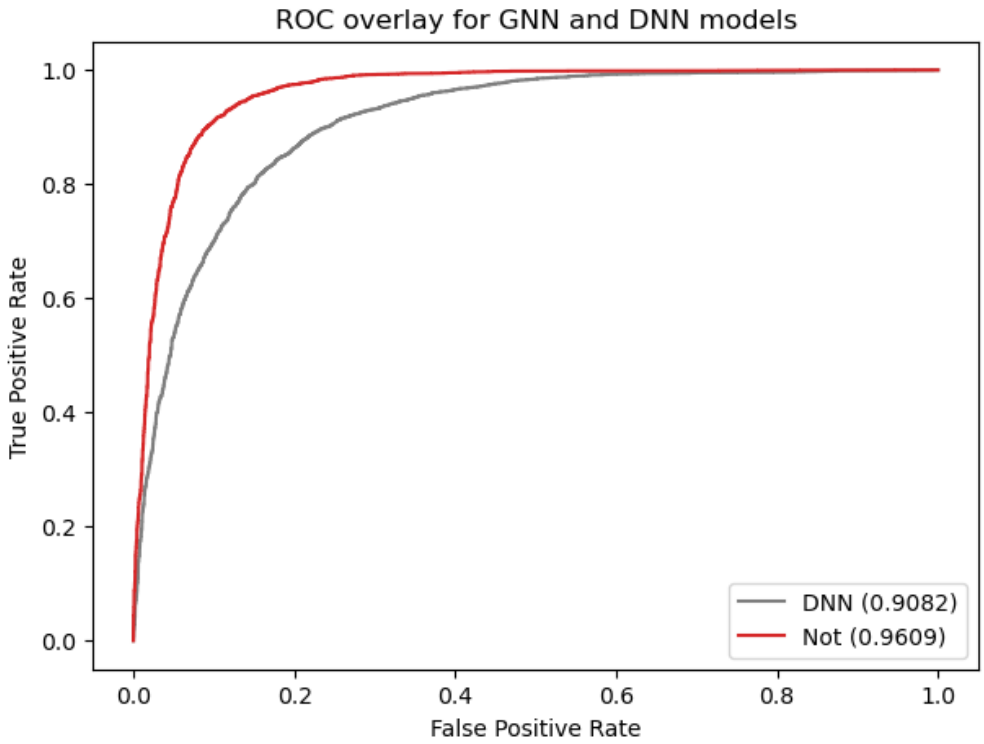


Figure 5.6: Comparison of test ROC curves for GNN and DNN models

Chapter 6

Impact of input graphs, GN layers and pooling in event classification

In Chapter 5, we established a baseline event classification framework using GNNs to differentiate H $\rightarrow$ ZZ events from ZZ. However, the choice of input graph structure, GNN architecture, pooling layer, and four-momentum representation impacts the model's performance. In this chapter, we systematically examine how different factors impact model performance, including:

1. **Graph structures**: Seven different input graph configurations.
2. **Graph convolution layers**: Comparison of GCNConv and GraphConv layers.
3. **Pooling methods**: Mean, Max, and Add pooling.
4. **Node features**: p_x, p_y, p_z, E vs p_{T}, η, ϕ, M as node features.

Additionally, we extend this comparative analysis to two new classification tasks:

1. $H \rightarrow WW$ versus WW event classification
2. ZZ versus WW event classification
3. $H \rightarrow ZZ$ versus $H \rightarrow WW$ event classification

This exercise gives us a good understanding of how different graph structures influence GNN performance, how GCNConv and GraphConv operators perform, the role of various pooling layers and how node feature representation affects the model outcomes. These findings serve as a foundation for future HEP analyses leveraging GNNs.

6.1 $H \to ZZ$ versus ZZ

We begin by revisiting the H → ZZ versus ZZ classification task, analyzing how different design choices impact model performance before extending our study to the two additional classification problems

6.1.1 Dataset

This section uses a fixed dataset of 10,000 events (same as 5.1) each for the $H \to ZZ$ and ZZ processes. We maintain the same training and testing splits for all models to ensure a fair comparison across different configurations.

6.1.2 Impact of graph structures

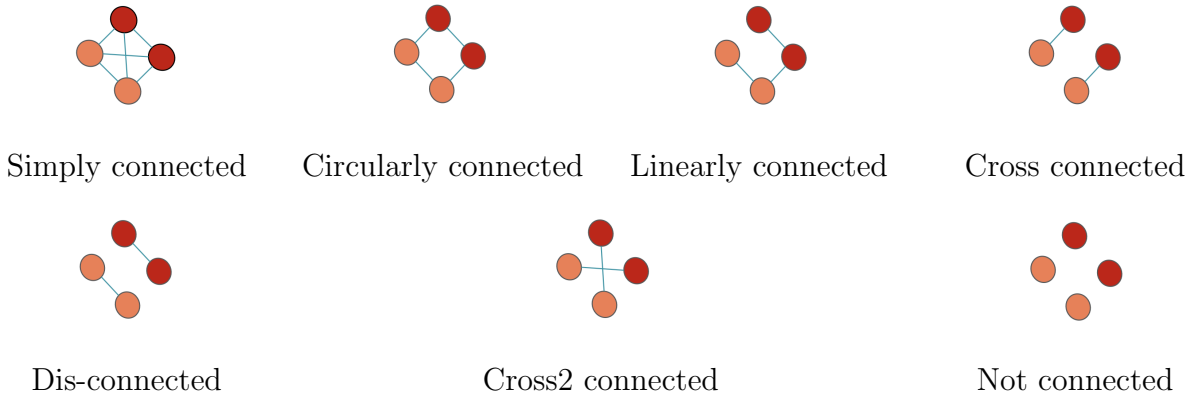


Figure 6.8: Few possible input graph structures with four nodes

We can construct various graph structures with these four nodes by defining different edge connections. For instance, we can fully connect all nodes, arrange them in a circular

or linear sequence, or establish connections based on shared parent particles. Figure 6.8 illustrates seven possible graph structures for our event classification task, along with the corresponding naming conventions we are using.

Figure 6.9 presents an overlay of the test ROC curves for models trained independently on each of the seven graph structures. Each curve represents the performance of a GNN model trained with a specific edge configuration, allowing us to compare how different graph constructions influence event classification. We use graphs with four momentum as node features, GCNConv layers and max pooling.

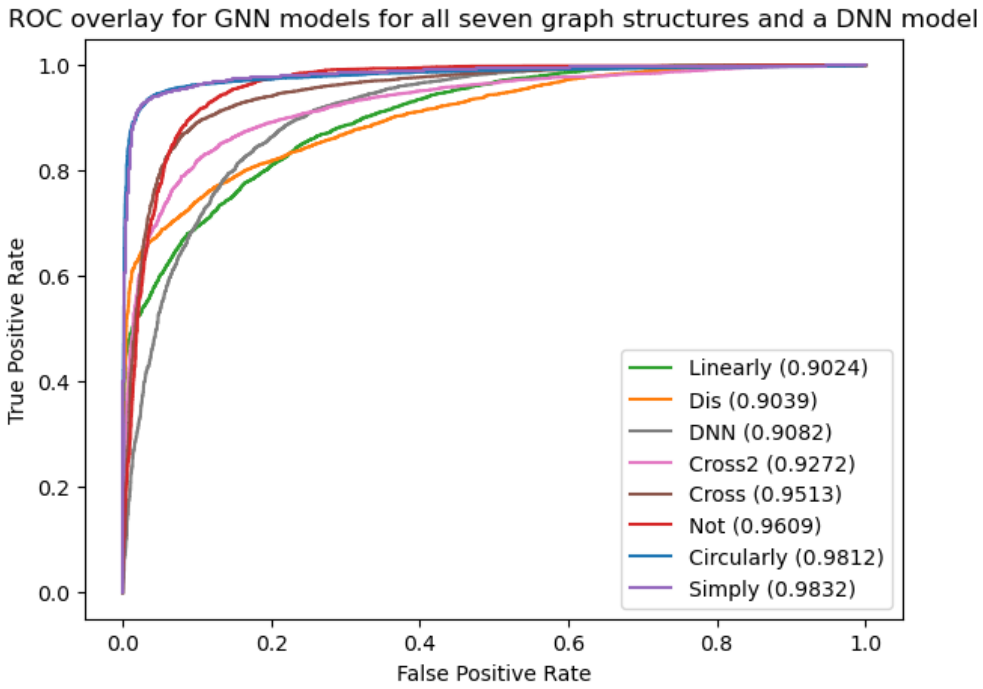


Figure 6.9: Overlay of ROC curves for $H \to ZZ$ versus ZZ GNN models for all seven graph structures and a DNN model

The area under the ROC curve (AUC) quantitatively measures each model's effectiveness in classifying events. The 'simply connected' graph comes on top, achieving the highest AUC of 98.3%

The 'cross' and 'cross2' structures are in the middle range, with AUCs of 95.1%

This confirms that leveraging graph-based representations can significantly boost performance, and choosing the proper graph structure makes a noticeable difference.

6.1.3 Impact of pooling methods

Pooling determines how node features are aggregated into a global graph representation. Therefore, another aspect is the choice of pooling method. We compare three commonly used pooling strategies: **Mean**, **Max**, and **Add** pooling.

While Figure 6.9 presents the ROC comparison for models with Max pooling, Figure 6.10 presents the overlays of ROC curves for models trained with Mean (left) and Add (right) pooling methods. Each plot includes all eight previously described ROC curves for a comprehensive comparison and validation.

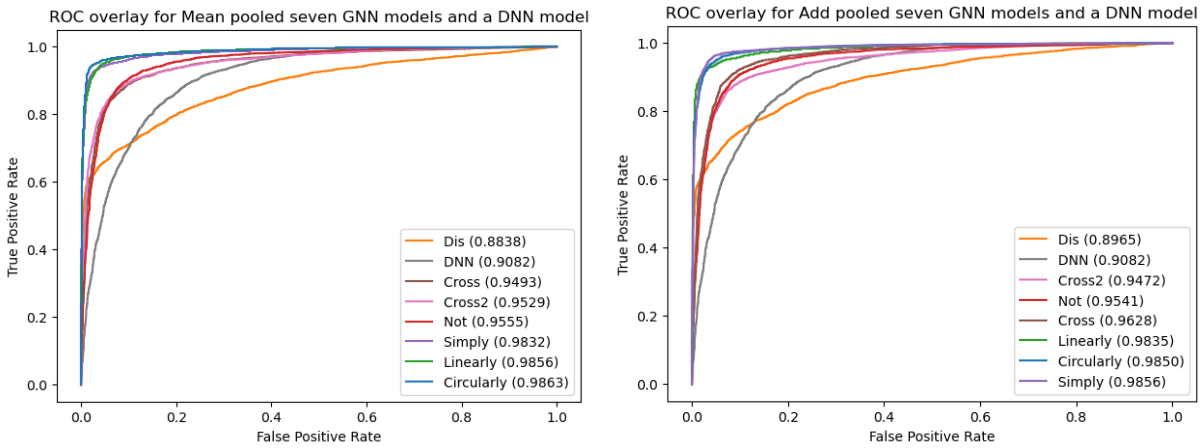


Figure 6.10: Overlay of ROC curves for $H \to ZZ$ versus ZZ with four momentum as features using Mean pooling (left) and Add pooling (right)

The results show that the choice of pooling method has minimal impact on classification performance. All three pooling strategies yield nearly identical AUC values across different graph structures, suggesting that graph connectivity influences event classification performance more than the specific pooling technique. However, one notable exception is the 'linearly connected' graph, which shows significant improvement with Mean and Add pooling compared to Max pooling, an effect that can be further explained.

6.1.4 Impact of graph convolution layers

Beyond graph structure and pooling methods, the choice of graph convolution operation also plays a crucial role in classification performance. Different convolution techniques define how

information is propagated and aggregated across the graph, potentially affecting the model's learning ability.

While Figure 6.9 compares models using **GCNConv**, Figure 6.11 presents the ROC curves for models trained with **GraphConv**. Each model is trained on the same dataset using all seven graph structures, allowing us to isolate the impact of convolution choice.

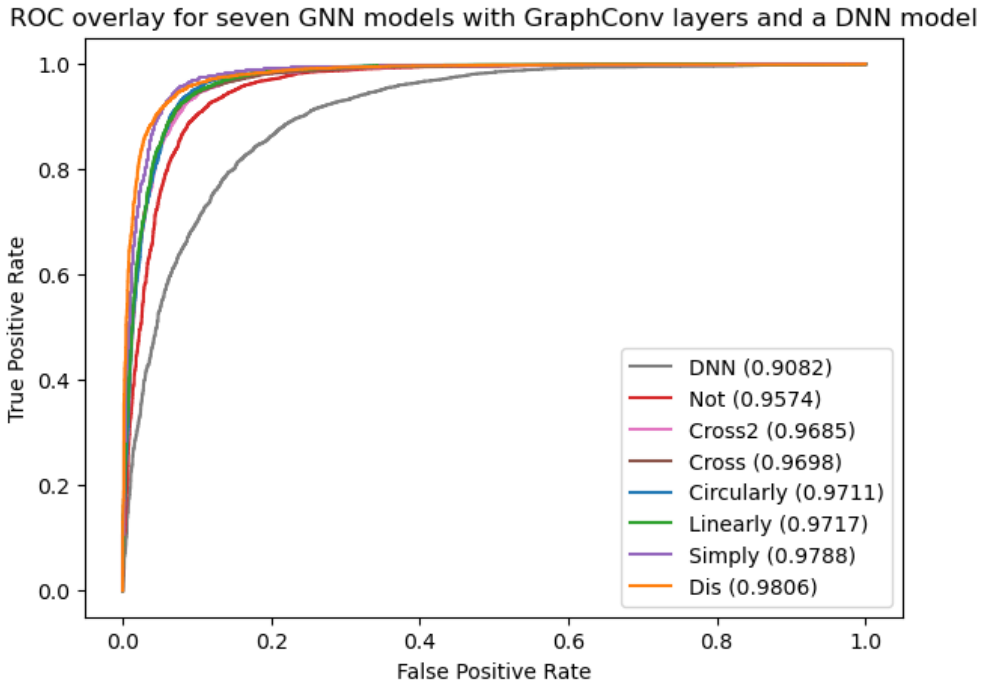


Figure 6.11: Overlay of ROC curves comparing H $\rightarrow$ ZZ vs ZZ using GraphConv architectures with Max pooling

The results indicate that the choice of graph convolution operation significantly impacts classification performance. An important consideration here is that the GCNConv model in Figure 6.9 is trained with a latent graph dimension of 8, resulting in 121 trainable parameters, whereas the GraphConv model in Figure 6.11 is trained with a latent dimension of 5, leading to 106 trainable parameters. This adjustment is necessary because using GraphConv with eight latent dimensions would increase the number of trainable parameters to 217, almost double that of GCNConv. To ensure a fair comparison, we reduce the complexity of GraphConv to match the parameter count of GCNConv.

Surprisingly, even after reducing the latent dimensionality and maintaining a similar number of trainable parameters, GraphConv still outperforms GCNConv. Not only does it

achieve better AUC values despite having 12% fewer parameters, but it also mitigates the impact of poor graph structures on performance. While the best-performing graph structures remain consistent between convolution types, the worst-performing structures degrade less in GraphConv than in GCNConv. This results in a narrower spread in the ROC overlays, indicating greater robustness across different graph structures.

Overall, GraphConv consistently outperforms both GCNConv and DNN, making it a more practical choice for this classification task.

6.1.5 Impact of node features

We must choose their node features carefully after deciding to use final-state particles as nodes. Previously, we evaluated model performance using four momentum (p_x, p_y, p_z, E) as node features. In this subsection, we explore the impact of using kinematic properties (p_T, η, ϕ, M) instead.

Figure 6.12 presents the distribution of kinematic properties for all four particles. The histograms in each row correspond to the components (p_T, η, ϕ, and M, from left to right) for different particles in the final state: the leading lepton, subleading lepton, leading neutrino, and subleading neutrino (from top to bottom). This follows the structure of Figure 5.1, which previously showed the distributions in four momentum features. Like Table 5.1, these node features are normalized using min-max scaling within predefined ranges. The maximum and minimum values for each feature are summarized in Table 6.1.

We compare the classification performance of both GCNConv and GraphConv models trained on kinematic properties node representations to evaluate the impact of using kinematic properties as node features instead of four momentum. Figures 6.13a and 6.13b present each case's ROC curves, respectively.

Comparing these results with those obtained using four momentum features (Figures 6.9 and 6.11) for GCNConv and GraphConv, respectively, we observe notable differences. It is essential to highlight that all GCNConv, GraphConv, and DNN share identically complex architectures and hyperparameters across both coordinate systems. The only variation lies in the input representation (four momentum versus kinematic properties), ensuring a fair comparison across all seven graph structures using MAX pooling.

Table 6.1: Normalization ranges for node features p_T, η, ϕ, and M

Particle	Feature	Range (Min, Max)
Leading Lepton	p_T	(0, 200)
	η	(-5, 5)
	ϕ	(-3.5, 3.5)
	M	(-0.15, 0.15)
Subleading Lepton	p_T	(0, 100)
	η	(-5, 5)
	ϕ	(-3.5, 3.5)
	M	(-0.15, 0.15)
Leading Lepton Neutrino	p_T	(0, 150)
	η	(-5, 5)
	ϕ	(-3.5, 3.5)
	M	(-0.15, 0.15)
Subleading Lepton Neutrino	p_T	(0, 75)
	η	(-5, 5)
	ϕ	(-3.5, 3.5)
	M	(-0.15, 0.15)

1. **GCNConv**: The classification performance of GCNConv declines when using kinematic properties as features (Figure 6.13a) compared to four momentum (Figure 6.9). While AUC values for four momentum features range between 0.90 and 0.98, the kinematic properties as node features results in a more constrained AUC distribution centred around 0.93 across all graph structures. This suggests that GCNConv struggles to leverage the kinematic properties coordinate system as effectively as the four momentum one.

2. **GraphConv**: In contrast, GraphConv shows good improvement in classification performance when trained with kinematic properties features (Figure 6.13b) over four momentum (Figure 6.11). Using kinematic properties as node features leads to higher AUC values, indicating better separation between signal and background events.

3. **DNN**: The DNN model shows a significant improvement in performance to 0.97 from 0.90 when using kinematic properties as features (Figure 6.13a) compared to four momentum (Figure 6.9). This shows DNN's capability to leverage kinematic properties as features effectively.

These results highlight the impact of coordinate representations on model performance.

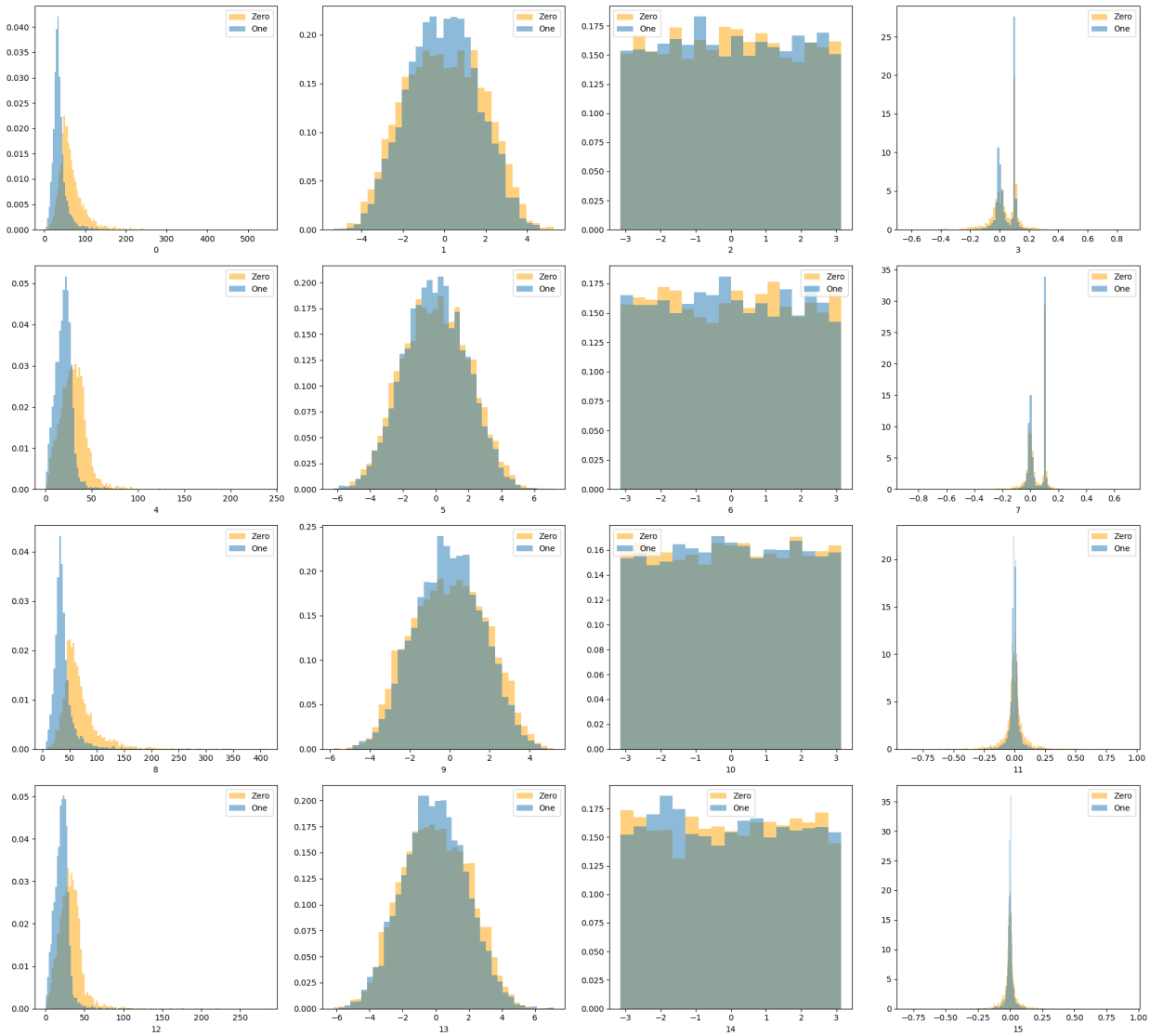
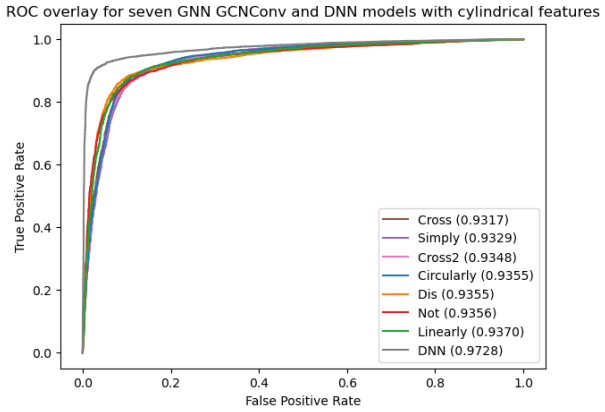


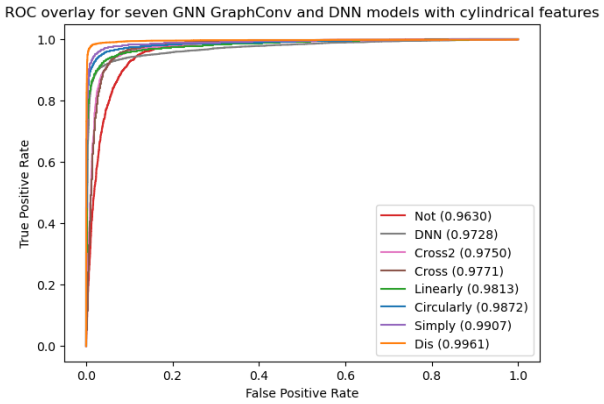
Figure 6.12: Horizontal axis: p_T, η, ϕ, M; vertical axis: all four particles sorted by p_T for both $H \to ZZ$ (one) and ZZ (zero) events

While GCNConv struggles to generalize well as kinematic properties, GraphConv benefits from the transformation, and the DNN shows the most significant improvement.

The following sections extend this analysis by performing similar variations for the classification tasks of H $\to$ WW versus WW, ZZ versus WW and H $\to$ ZZ versus H $\to$ WW.



(a) GCNConv with kinematic properties as features



(b) GraphConv with kinematic properties as features

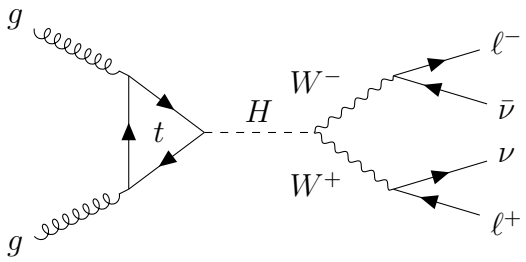
Figure 6.13: Overlay of ROC curves comparing H → ZZ vs ZZ with kinematic properties as features using GCNConv (left) and GraphConv (right) architectures with Max pooling

6.2 $H \to WW$ versus WW

After analyzing the H → ZZ versus ZZ classification, I present the results for the H → WW versus WW classification. This task distinguishes Higgs boson decaying into two W bosons ($H \to WW$) and non-resonant diboson WW production (background). This classification problem is structurally identical to the previous one, with the only difference being the replacement of Z bosons with W bosons. The primary motivation for this analysis was to validate the impact of graph structures, convolution layers and pooling layers from the H → ZZ versus ZZ classification.

6.2.1 Dataset and Preprocessing

Here, the signal process is:

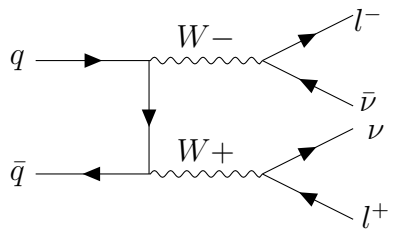


(6.1)

$$H \to WW \to \ell^-\ell^+\nu\bar{\nu}$$

In this process, p-p collisions produce a Higgs boson (H), which subsequently decays into two W bosons. Both the W bosons decay leptonically into a charged lepton and corresponding neutrino ($\ell^+\nu$ or $\ell^-\bar{\nu}$, where ℓ represents either an electron or a muon).

The background process is:



(6.2)

$$W^+W^- \to \ell^-\ell^+\nu\bar{\nu}$$

Here, the p-p collision produces two W bosons. Both the W bosons decay leptonically into a charged lepton and corresponding neutrino ($\ell^+\nu$ or $\ell^-\bar{\nu}$, where ℓ represents either an electron or a muon) leading to the same final state as the signal: $\ell^-\ell^+\nu\bar{\nu}$.

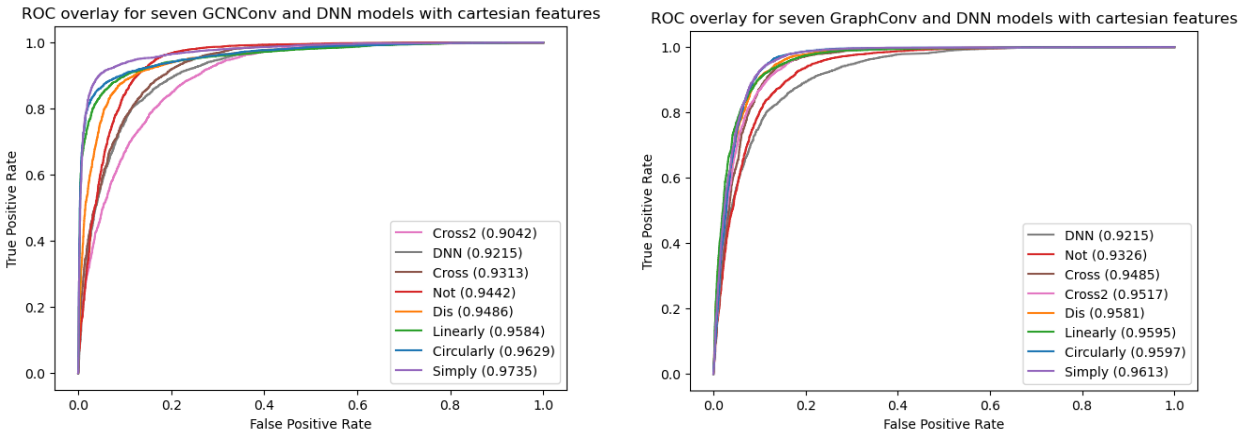
Similar to the previous classification task, the dataset consists of 10,000 Monte Carlo simulated events for each process. These events are evenly split into a **training** and **testing** dataset, with 5,000 signal events and 5,000 background events in each subset.

6.2.2 Results and Discussion

As done in the previous section, we present ROC overlay plots for GNN models with all seven graph structures and a standard DNN. Each comparison is made for GCNConv and GraphConv architectures to highlight performance differences, allowing us to assess the impacts of graph structures and convolutional layers systematically. For this classification we are focusing only on the four momentum node features.

Impact of graph convolution layers

Figures 6.14a and 6.14b show the overlays of ROC curves for seven graph structures and DNN for GCNConv and GraphConv, respectively. These results are for GNN architectures with Max pooling. It is clear from both examples that for event classification, where one process has all final state particles coming from a single particle versus another process where they are not simply connected or circularly connected, graph structures perform better than the rest. GraphConv performs better than GCNConv, considering the reduced latent dimensionality.



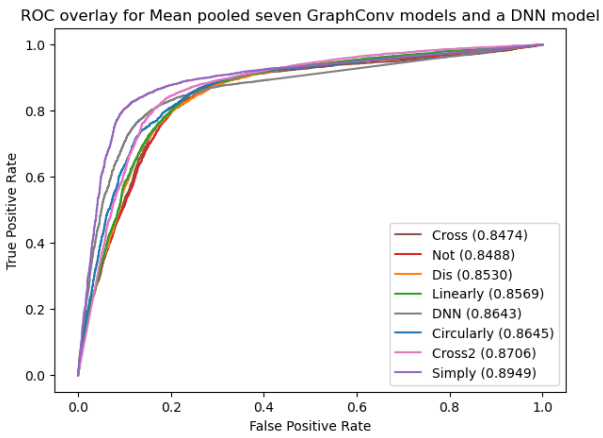
(a) GCNConv with four momentum features (b) GraphConv with four momentum features

Figure 6.14: Overlay of ROC curves comparing $H \rightarrow WW$ vs WW in four momentum features using GCNConv (left) and GraphConv (right) architectures with Max pooling

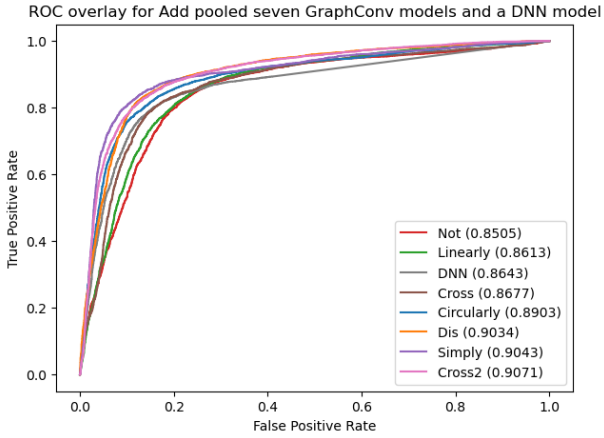
Mean and Add pooling

Next, we examine the impact of different pooling mechanisms.

Figures 6.15a and 6.15b present the corresponding Mean and Add pooling strategies results. These are the results for GraphConv layers. It is clear from the results that the pooling layer has little to no effect on the classification outcome, so moving forward, we only focus on the Max pooling.



(a) GraphConv with four momentum features and Mean pooling



(b) GraphConv with four momentum features and Add pooling

Figure 6.15: Overlay of ROC curves for $H \to WW$ versus WW in four momentum features using Mean pooling (left) and Add pooling (right)

6.3 ZZ versus WW

After analyzing the H $\to$ ZZ versus ZZ and H $\to$ WW versus WW classification, I present the results for the ZZ versus WW classification. In this task, we distinguish between events with non-resonant diboson productions ZZ versus WW. This is an interesting classification as both processes' signal and background do not include Higgs decay, unlike previous tasks where the signal process had Higgs production. The primary motivation for this analysis was to check the impact of various graph structures and apply and test the performance of GATConv layers, which use the graph attention mechanism.

6.3.1 Dataset and Preprocessing

Here, the signal process is

$$ZZ \to \ell^-\ell^+\nu\bar{\nu}$$

same as process 5.3

The background process is

$$W^+W^- \to \ell^-\ell^+\nu\bar{\nu}$$

same as process 6.2

Like the previous classification tasks, the dataset consists of 10,000 Monte Carlo simulated events for each process. These events are evenly split into a **training** and **testing** dataset, with 5,000 signal events and 5,000 background events in each subset.

6.3.2 Results and Discussion

We present ROC overlay plots for GNN models with all seven graph structures and a standard DNN. Each comparison is made for GCNConv and GraphConv architectures to highlight performance differences, allowing us to assess the impacts of graph structures and convolutional layers systematically. We evaluate the impact of node feature representation by considering both four momentum and kinematic properties as features.

Along with the usual comparison, we test GATConv layers that utilize the graph attention mechanism. As Chapter 4.5.1 mentions, the attention mechanism decides which nodes are more important to focus on.

Four momentum node features

Let's start with four momentum node feature representation,

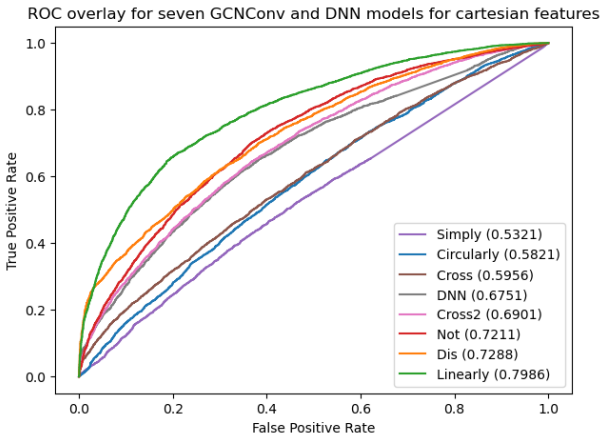
Figures 6.16a and 6.16b show the ROC curves for GCNConv and Graphconv cases, respectively. These are the results for GNN architectures with Max pooling.

Attention mechanism

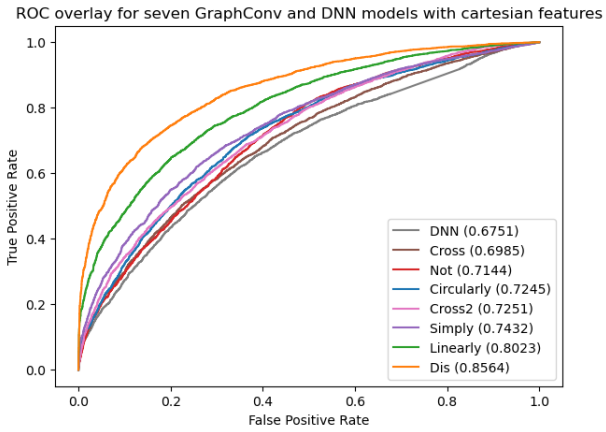
Here, we use the GATConv layer for the attention mechanism with single and double-headed attention for the graphs with four momentum features.

Figures 6.17 and 6.17 show the ROC curves for GAT architectures with Max pooling for single-headed attention and double-headed attention, respectively.

It is clear from the results that the graph attention mechanism is also very much de-

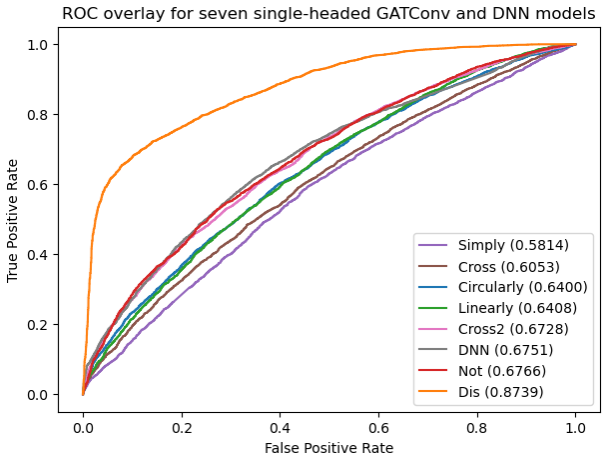


(a) GCNConv with four momentum features do

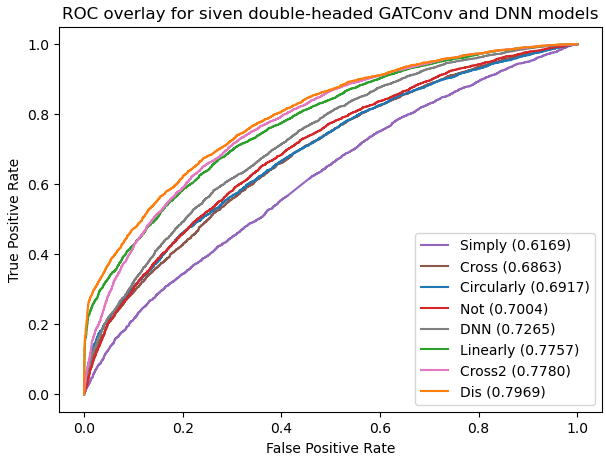


(b) GraphConv with four momentum features do

Figure 6.16: Overlay of ROC curves comparing ZZ vs WW in four momentum features using GCNConv (left) and GraphConv (right) architectures with Max pooling



(a) GATConv with single-headed attention



(b) GATConv with double-headed attention

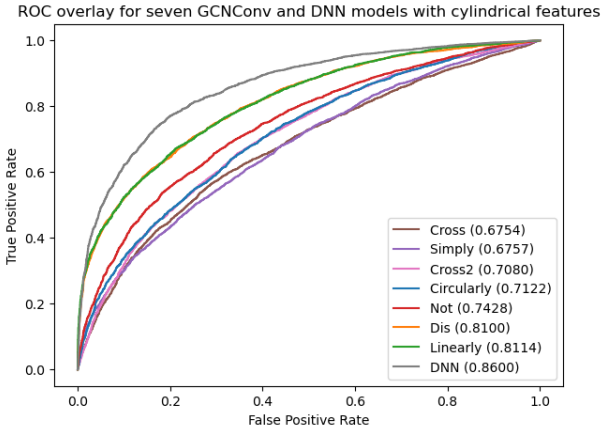
Figure 6.17: Overlay of ROC curves comparing ZZ vs WW in four momentum features using GATConv single-headed attention (left) and double-headed attention (right) architectures with Max pooling

pendent on the input graph structures. Although single-headed attention for disconnected graphs is doing slightly better (AUC = 0.87) than for GraphConv (AUC = 0.85, from Figure 6.16b), the overall attention mechanism is not helping the event classification problem.

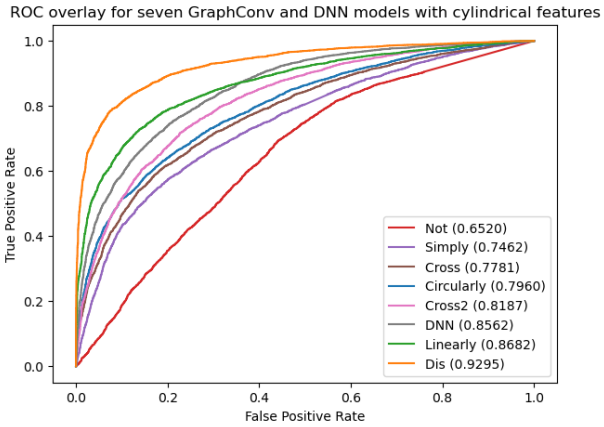
kinematic properties as node features

Now, for kinematic properties as node feature representation,

Figures 6.18a and 6.18b show the ROC curves for GCNConv and Graphconv cases, respectively. These are the results for GNN architectures with Max pooling. As seen before, kinematic properties as node features improve the performance over four momentum node features.



(a) GCNConv with kinematic properties as features



(b) GraphConv with kinematic properties as features

Figure 6.18: Overlay of ROC curves comparing ZZ vs WW in kinematic properties as features using GCNConv (left) and GraphConv (right) architectures with Max pooling

6.4 $H \to ZZ$ versus $H \to WW$

After analyzing the H $\to$ ZZ versus ZZ, H $\to$ WW versus WW and ZZ versus WW classification, I now present the results for the H $\to$ ZZ versus H $\to$ WW classification. In this task, we distinguish between events with Higgs bosons decaying into two Z bosons ($H \to ZZ$) or two W bosons ($H \to WW$). This is also an interesting classification as the signal and background include processes with Higgs, whereas previous tasks were against the non-Higgs background. The primary motivation for this analysis was to identify which graph structures should perform better before trying out all seven.

6.4.1 Dataset and Preprocessing

Here, the signal process is

$$H \to ZZ \to \ell^-\ell^+\nu\bar{\nu}$$

same as process 5.2

The background process is

$$H \to WW \to \ell^-\ell^+\nu\bar{\nu}$$

same as process 6.1

Like the previous classification tasks, the dataset consists of 10,000 Monte Carlo simulated events for each process. These events are evenly split into a **training** and **testing** dataset, with 5,000 signal events and 5,000 background events in each subset.

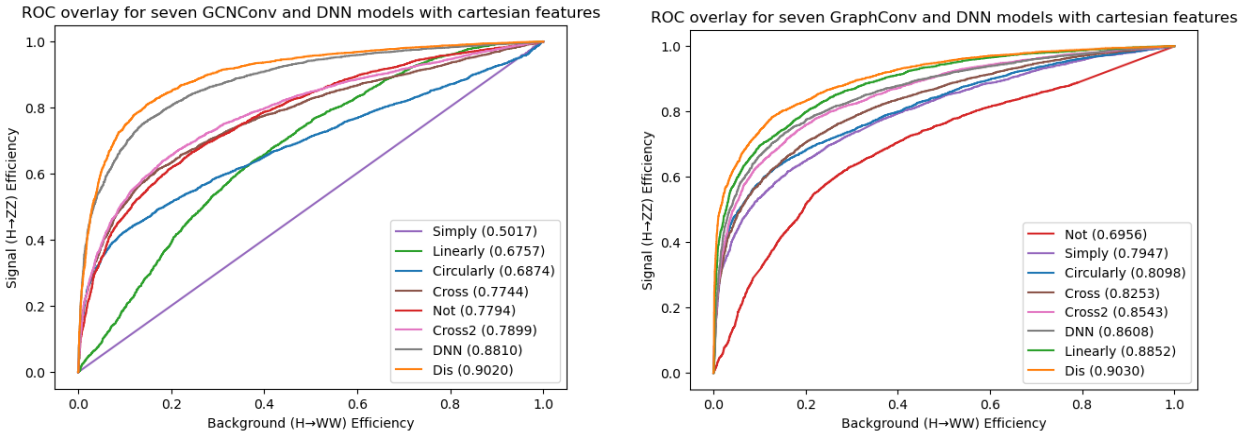
6.4.2 Results and Discussion

We present ROC overlay plots for GNN models with all seven graph structures and a standard DNN. Each comparison is made for GCNConv and GraphConv architectures to highlight performance differences, allowing us to assess the impacts of graph structures and convolutional layers systematically. We evaluate the impact of node feature representation by considering both four momentum and kinematic properties as features. Here, as we learned from previous examples, for signal processes involving Higgs and background not involving Higgs, simply connected or circularly connected graphs do better. But if both processes don't include Higgs, they do poorly, and dis-connected graphs do better. So, we expect a similar trend for both processes involving Higgs, where simply connected graphs would do poorly. Let's check our hypothesis.

Four momentum node features

Let's start with four momentum node feature representation,

Figures 6.19a and 6.19b show the ROC curves for GCNConv and Graphconv cases, respectively. These are the results for GNN architectures with Max pooling.



(a) GCNConv with four momentum features (b) GraphConv with four momentum features

Figure 6.19: Overlay of ROC curves comparing H $\rightarrow$ ZZ vs H $\rightarrow$ WW in four momentum features using GCNConv (left) and GraphConv (right) architectures with Max pooling

Observing Figures 6.19a, we see that the simply connected graph yields an AUC of 0.50, equivalent to random classification. This indicates that for H $\rightarrow$ ZZ versus H $\rightarrow$ WW classification, the GCNConv model fails to learn any meaningful patterns from the simply connected graph. In contrast, the dis-connected graph achieves an AUC of 0.90, outperforming the DNN model. Notably, both models use the same GNN architecture with 385 trainable parameters, meaning the only difference is the input graph structure. This demonstrates that an appropriate graph structure can significantly enhance performance, enabling a GNN to outperform DNNs, whereas a poor structure can hinder learning completely.

Figures 6.19b show the same classification task using the GraphConv model, which has only 217 trainable parameters, nearly half that of the GCNConv model. Despite the reduced parameter count, GraphConv achieves comparable performance for dis-connected graphs and effectively mitigates the learning failure observed with simply connected graphs, leading to improved overall performance.

Figure 6.20 further illustrates how input graph structures influence model behaviour. The classification score distribution reveals that simply connected graphs fail to distinguish between classes, resulting in a sharply peaked output. In contrast, dis-connected graphs exhibit a well-separated distribution, indicating superior classification capability. The middle section of the figure explains the learning dynamics: in simply connected graphs, every node interacts with every other node, whereas in dis-connected graphs, only leptons interact with

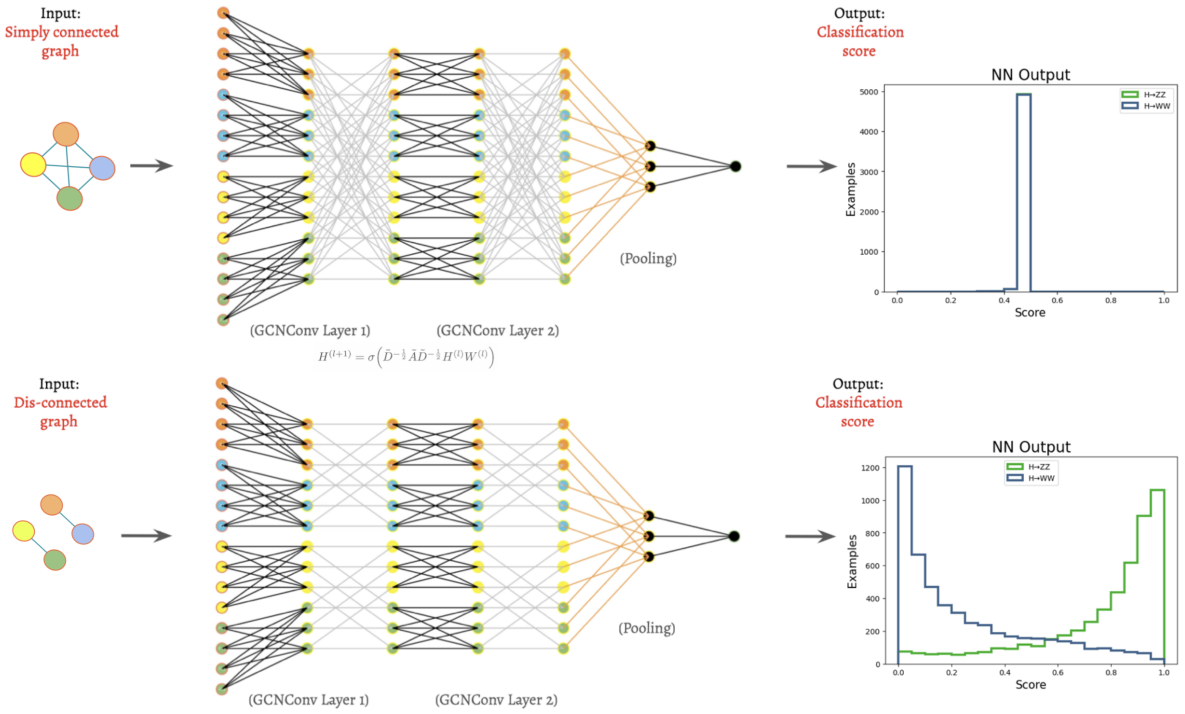


Figure 6.20: Simply connected graphs vs dis-connected graphs

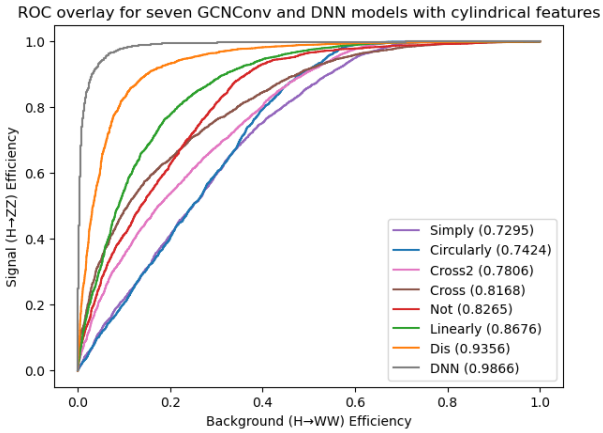
each other, and only neutrinos interact with each other. This structural distinction is crucial: simply connected graphs fail to differentiate between signal and background processes, while dis-connected graphs enable good classification.

Dis-connected graphs consistently achieve the highest performance for this classification, followed by DNN models, whereas simply connected and circularly connected graphs underperform. This tells us that simply connected graphs are the worst choice when a single particle decays into all final-state objects. These results emphasize the critical role of graph structure in GNN-based classification—poor connectivity choices can severely hinder learning, whereas well-designed graphs can significantly enhance model performance.

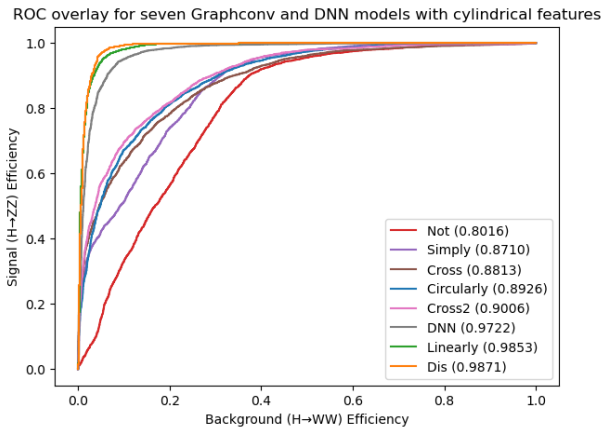
kinematic properties as node features

Now, for kinematic properties as node feature representation,

Figures 6.21a and 6.21b show the ROC curves for GCNConv and GraphConv cases,



(a) GCNConv with kinematic properties as features



(b) GraphConv with kinematic properties as features

Figure 6.21: Overlay of ROC curves comparing H $\rightarrow$ ZZ vs H $\rightarrow$ WW with kinematic properties as features using GCNConv (left) and GraphConv (right) architectures with Max pooling

respectively. These are the results for GNN architectures with Max pooling.

Here, we see that the GraphConv model with kinematic properties as node features and Max pooling for all event classification tasks shows the best performance. The choice of graph structure depends on the classification problem.

6.5 MyConv

GNN models have shown significant improvement in performance for event classification tasks. However, In standard GNNs, edge weights are either fixed (typically 0 or 1) or implicitly learned through node aggregation functions, which may limit the model's ability to classify between events. Inspired by the attention mechanism used in graph attention networks (GATConv), I have developed a novel GNN architecture called **MyConv**, which incorporates learnable edge weights to enhance the model's ability to differentiate between signal and background events. The motivation behind MyConv is to allow the model to learn edge weights dynamically based on the input graph structure and node features, enabling the network to capture relational patterns.

6.5.1 Methodology

The MyConv layer builds upon the concept of attention-based aggregation in GATConv layers but introduces an explicit learnable edge weight mechanism. The forward pass of MyConv can be summarized as follows:

6.5.2 Model Architecture

The MyConv model consists of multiple convolutional layers equipped with trainable edge weights. The forward pass of MyConv is described as follows:

1. **Input and Initialization:** The model receives a graph input consisting of node features $\mathbf{X} \in \mathbb{R}^{N \times F}$ and edge indices $\mathbf{E} \in \mathbb{R}^{2 \times M}$, where N is the number of nodes, F is the number of node features, and M is the number of edges.

2. **Learnable Edge Weights:** A trainable parameter vector $\mathbf{w} \in \mathbb{R}^{M}$ is initialized randomly and optimized during training. The edge weights are computed using a sigmoid activation to constrain them within the range $(0, 1)$:

$$w_{ij} = \sigma(\mathbf{w}_{ij}) \tag{6.3}$$

   where σ is the sigmoid function.

3. **Convolutional Layers:** The node features are processed using a series of convolutional layers, where each layer incorporates the learned edge weights to adjust the message-passing step:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} w_{ij} \cdot \mathbf{W}^{(l)} \mathbf{h}_j^{(l)}\right) \tag{6.4}$$

   where $\mathbf{W}^{(l)}$ is the learnable weight matrix of the l-th convolutional layer and $\mathcal{N}(i)$ denotes the neighbours of node i.

4. **Pooling and Classification:** After the convolutional layers, the output node embeddings are pooled into a single graph representation using a global Max pooling operator:

$$\mathbf{z} = \text{Max}(\mathbf{h}^{(L)}) \tag{6.5}$$

where $\mathbf{h}^{(L)}$ are the node embeddings from the final layer. The pooled output is passed through a linear layer followed by a sigmoid activation for binary classification:

$$\hat{y} = \sigma(\mathbf{W}_{\text{out}}\mathbf{z} + b) \tag{6.6}$$

This approach allows the model to learn node-wise embeddings and edge weights simultaneously, enhancing the network's ability to capture event topology.

6.5.3 Results and Performance

To evaluate the performance of MyConv, I trained the model on the $H \to ZZ$ versus ZZ classification task using binary cross-entropy (BCE) loss.

The ROC curve in Fig. 6.22 summarizes the results, comparing MyConv's performance under different graph input structures with a DNN baseline.

MyConv demonstrates significant improvements over traditional DNN approaches. The model achieves the highest AUC (0.91) for simply connected, circularly connected and linearly connected input graphs. Compared to the DNN baseline (AUC = 0.8608), MyConv's ability to leverage learnable edge weights leads to more effective signal-background separation.

This indicates that MyConv can adapt effectively to different graph structures. This adaptability mitigates the drastic impact of various graph structures on GNN performance over fixed structured data representations in HEP analysis. MyConv is designed to handle graph inputs with varying numbers of edges. The model dynamically adjusts to the number of edges provided in the input graph and learns edge weights specifically for the given graph structure, allowing for greater flexibility in handling different graph configurations.

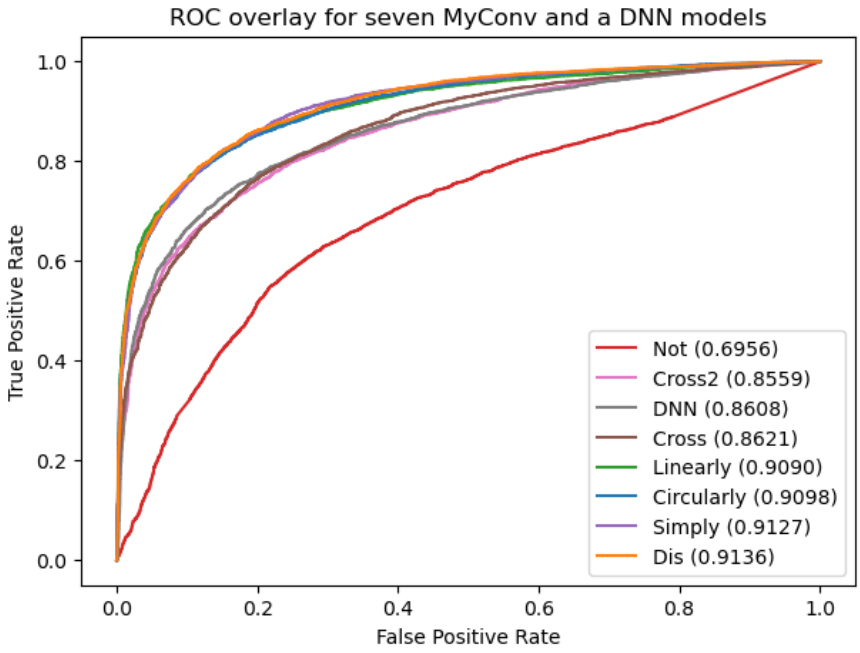


Figure 6.22: Overlay of ROC curves comparing H → ZZ vs H → WW using MyConv model

The following Table 6.2 shows the comparison of classification accuracy between MyConv and other used GNN models for dis-connected and simply connected graphs:

Model	**Dis-connected AUC (%)**	**Simply connected AUC (%)**
GCNConv	90.2	50.17
GraphConv	90.3	69.56
MyConv	**91.36**	**91.27**

Table 6.2: Comparison of classification performance between different GNN models for the $H \to ZZ$ versus $H \to WW$ task.

MyConv outperformed other GNN models, achieving a classification accuracy of 91%

6.5.4 Interpretability and Physical Insight

Figure 6.23 shows the simply connected graph before and after learning edge weights β_{ij}.

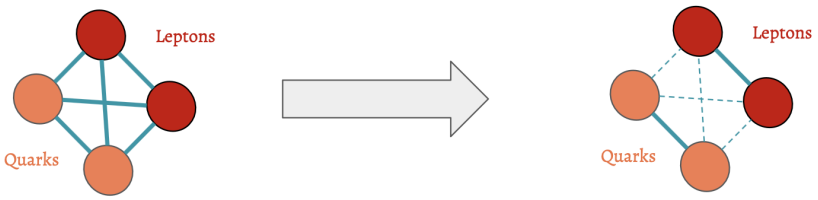


Figure 6.23: Simply connected before (left) and after learning weights by MyConv (right)

An analysis of the learned edge weights β_{ij} reveals that MyConv assigns higher weights to edges connecting two leptons and two neutrinos, consistent with the dis-connected graphs. This indicates that MyConv modifies simply connected graphs into dis-connected-like structures, demonstrating its potential to mitigate the impact of the input graph structure.

In conclusion, for any event classification problem, one can use MyConv layers on simply connected graphs instead of trying out all the possible graph structures and various GNN architectures.

Chapter 7

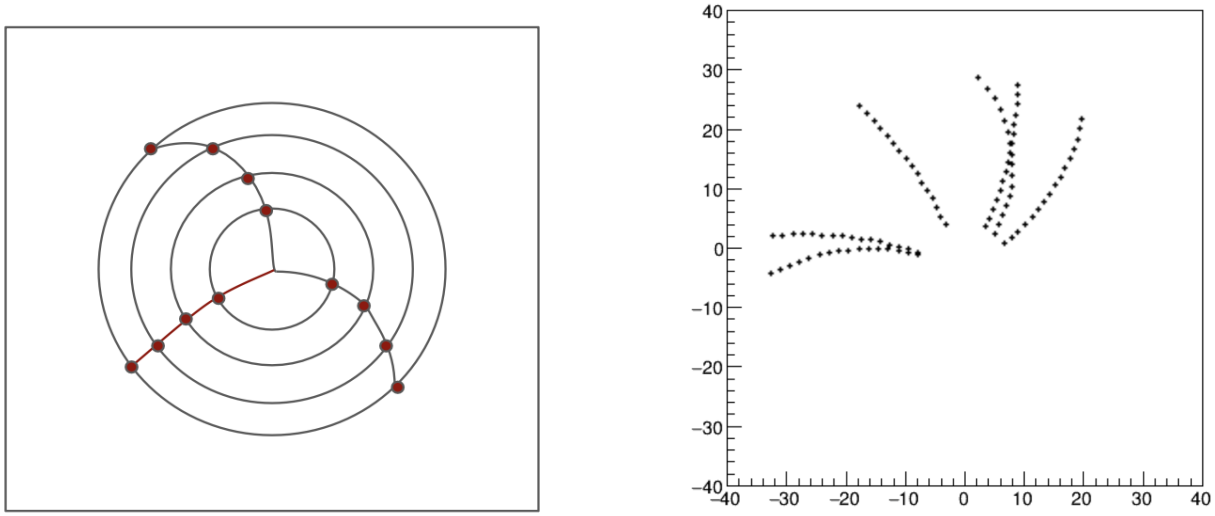
Tracking task

In LHC detectors, the track is the path a charged particle traces as it moves through the detector's magnetic field. These particles ionize the detector material, giving us hits in each layer. These signals are used to reconstruct the trajectory. The transverse momentum (p_{T}) of the particles influences the curvature of the tracks: particles with higher p_{T} follow straighter paths, while those with lower p_{T} curve more sharply due to stronger deflection by the magnetic field.

In this section, I focus on object-level analysis, specifically analyzing tracks. I am using a toy model that generates proxy tracker hits, simulating the low-level detector data corresponding to hits in the tracker. Now, the goal is to analyze these hits and classify whether a high-p_{T} particle is present in the event. For this we use GNNs with tracker hits forming a graph and using similar graph classification algorithms to classify the high-p_{T} events (signal) from low-p_{T} (background).

7.1 Toy tracker simulation

In the toy tracker simulation, we generate the specified number of charged particles in random directions in 4 Tesla magnetic field. This gives out all hits with their (x, y) coordinates. Figure 7.1a illustrates tracker hits for three tracks, with the red track being the high-p_{T} track. Figure 7.1b shows the actual simulation data for 6 (with two high-p_{T}) tracks.



(a) Illustration of tracker hits

(b) Actual tracker data for six-track events

Figure 7.1: An illustrative representation of tracker hits (a) with actual tracker data for a six-track event (b).

The toy tracker simulation is designed to mimic the response of a tracking detector in a 4 Tesla magnetic field. It takes the following parameters as input:

1. The number of circular detector layers in the tracker.

2. The number of high-p_{T} charged particles.

3. The number of low-p_{T} charged particles.

Using these inputs, the simulation generates particle tracks and computes the x and y coordinates of the corresponding hits in each layer. The curvature of each track is determined by the particle's p_{T}.

The final output consists of the total number of hits in the tracker, along with their 2D coordinates. This information is then transformed into different graphs for further analysis using GNNs.

7.2 Graph representation

In this context, the advantage of using GNN over DNN is the ability to handle variable-sized input data, as the number of particles and corresponding hits in the detector can vary from event to event. DNNs require fixed input dimensions, whereas GNNs can dynamically process graphs of different sizes, making them well-suited for analyzing detector data with varying complexity.

There are multiple ways to construct graphs from the coordinate data obtained from the toy tracker simulation. Each approach influences how the GNN learns spatial relationships and extracts meaningful features from the hit information.

1. **Point cloud graphs (No edges)**
    - The simplest representation treats each hit as an independent node.
    - The graph consists only of nodes with (x, y) coordinates as node features, but no edges are defined between them.
    - This setup allows the model to learn from individual hit positions without explicitly encoding spatial relationships between hits.
2. **Nearest neighbour graphs**
    - Nodes (hits) are connected to their nearest nodes in adjacent tracker layers.
    - This structure captures local geometric correlations, allowing the model to infer track curvature based on proximity.
3. **Distance as nodes**
    - Here, the nodes represent the distances between the two nearest hits in adjacent tracker layers.
    - The node features consist of the distance and the angle between the two nearest hits of the tracker layers.
    - This approach transforms track segment relationships into node-level features rather than edge-level connections, allowing the model to learn from the track geometry differently.

7.3 Results and discussions

7.3.1 Point cloud graphs

For this task, I trained my models on five datasets, each containing various number of tracks. The goal is to classify events based on the presence of at least one high-p_{T} track. Here, tracks are categorized as follows:

- **High-p_{T} track:** A charged particle with p_{T} in the range of $100 \leq p_{\mathrm{T}} \leq 150$ GeV.
- **Low-p_{T} track:** A charged particle with p_{T} in the range of $10 \leq p_{\mathrm{T}} \leq 50$ GeV.

The datasets are defined as follows:

1. Single-track dataset: One high-p_{T} track (signal) versus one low-p_{T} track (background).
2. Two-track dataset: An event containing one high-p_{T} track and a low-p_{T}(signal) versus two low-p_{T} track (background). This introduces an additional track compared to the single-track case.
3. Three-track dataset: Events containing three tracks, where at most one is a high-p_{T} track (signal) versus three low-p_{T} tracks(background).
4. Four-track dataset: Similar to the three-track dataset but with four tracks in the event, increasing complexity.
5. Six-track dataset with two high-p_{T} tracks: Events with six tracks, including two high-p_{T} tracks. This dataset introduces additional high-p_{T} track for signal.

These datasets allow the model to learn how well it can identify high-p_{T} tracks in increasingly complex scenarios, testing its ability to generalize across different events. Each dataset contains 10K training events and 10K testing events equally split between signal and background.

7.3.2 ROC Curve Analysis

GNN architecture includes four GCNConv layers with max pooling. The ROC curve for different datasets is shown in Figure 7.2.

1. The AUC is highest for the single-track dataset (1 ROC, AUC = 1), indicating perfect classification.
2. As the number of tracks increases, the classification becomes more challenging, leading to lower AUC values.
3. The model achieves AUC = 0.80 for two-track events, showing strong performance.
4. The three-track and four-track datasets yield AUC values of 0.66 and 0.58, respectively, highlighting the increasing difficulty in identifying high-p_{T} tracks as complexity grows.
5. The six-track dataset with two high-p_{T} tracks achieves AUC = 0.67.

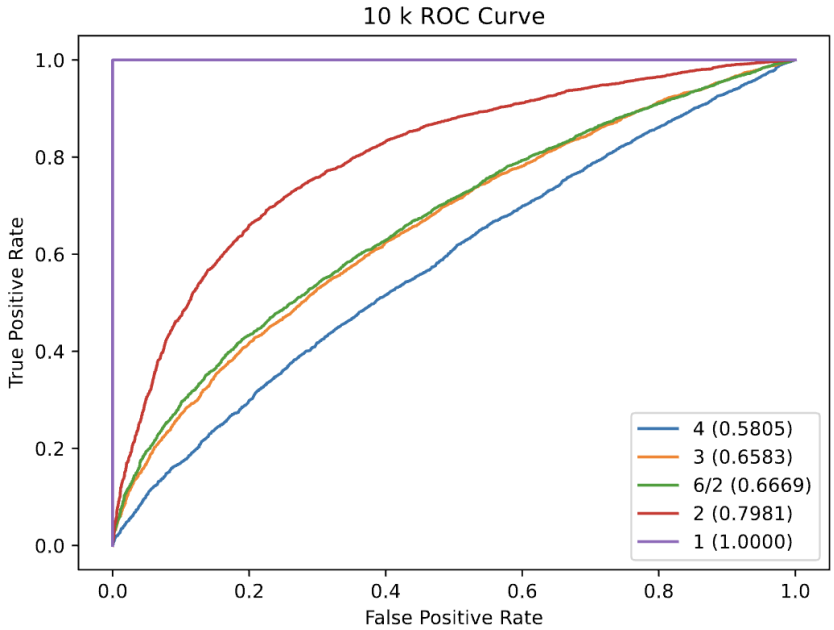


Figure 7.2: ROC curves showing model performance across increasing event complexity

These results suggest that while the model effectively distinguishes high-p_{T} tracks in simple scenarios, but further improvements are needed for events with higher track multiplicities.

7.3.3 Cross performance

One of the advantages of using GNNs is their ability to handle variable input sizes, allowing models trained on one dataset to be tested on datasets with different numbers of tracks. This flexibility enables us to investigate the generalization capability of our models across varying track multiplicities.

Figure 7.3 presents the results of testing models trained on datasets with a specific number of tracks on datasets with different track multiplicities. The key observations include a model trained on the three-track dataset tested on the six-track dataset with two high-p_{T} tracks, achieving an AUC of 0.64. This suggests the model retains reasonable performance even when tested on a more complex dataset. The model trained on two-track events was tested on the three and four-track datasets, achieving an AUC of 0.64 and 0.70, which is more than 0.58 and 0.66 from Figure 7.2. This indicates that the model can generalize moderately well to events with additional tracks.

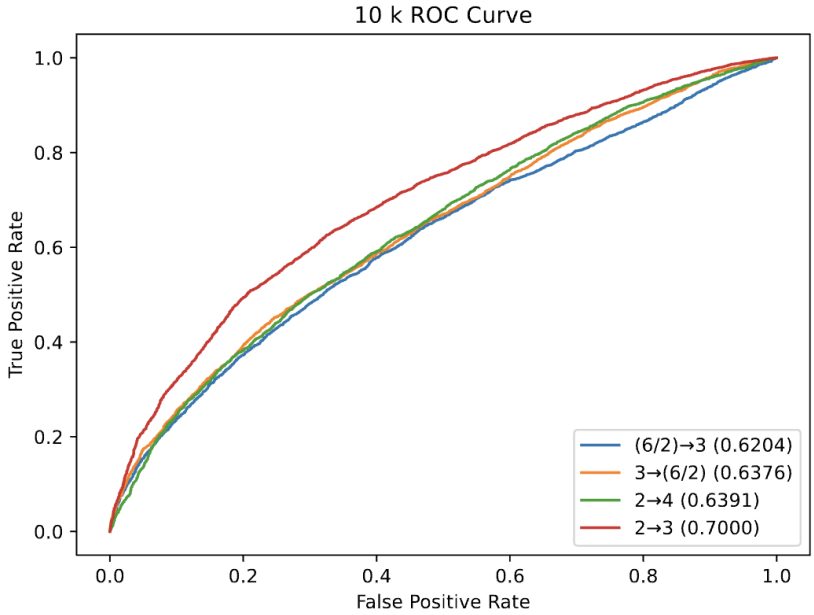


Figure 7.3: ROC curves for models trained on datasets with a specific number of tracks and tested on datasets with different track multiplicities

Chapter 8

Conclusion

In this thesis, I have explored the applications of GNNs in experimental HEP, particularly in the context of event classification and track-based analysis. The primary motivation for this study was to leverage the inductive biases of graph-based ML techniques for HEP analysis, which requires transforming HEP data into meaningful graph structures. Since no ready-to-use graph datasets are available for HEP data, I developed a Python-based workflow to generate custom HEP graph datasets. I have demonstrated the potential advantages and challenges of GNN techniques over other deep learning methods, such as DNNs, in analyzing LHC data. GNN presents a flexible framework for modelling collider events and detector outputs, making it a valuable tool for HEP analyses.

I designed and implemented multiple GNN-based frameworks for event classification to distinguish signal events from background events. Optimized GNN models consistently outperformed DNN models with similar trainable parameters. However, the performance of GNN models varied significantly depending on the design choices and architectural details. The choice of input graph structure, node feature representation, pooling methods, and GN layers influenced model performance.

For GNN models with `GCNConv`, `GraphConv` or `GATConv` layers, the input graph structure played a crucial role. Different graph structures performed better for different classification tasks. At the same time, some graph structures led to failed learning despite using identical GNN models. This highlights the challenge of selecting the optimal graph representation for HEP events.

To address this issue, I developed a novel GN layer, `MyConv`, that dynamically learns edge weights based on the input graph structure and node features. By incorporating learnable edge weights, `MyConv` enables the model to adapt to different graph structures, reducing the dependency of model performance on the input graph design and improving overall performance.

Through systematic investigation, we observed several key insights regarding the impacts of choices of input graph structures, node feature representations, pooling methods, and GNN architectures in event classification:

Impact of input graph structures

1. The choice of input graph structure significantly impacts model performance.

2. Poorly designed graph structures can hinder training completely.

3. Simply connected and circularly connected graphs tend to perform well in tasks like distinguishing $H \to ZZ$ events from ZZ background events or $H \to WW$ events from WW background events.

4. In contrast, for $H \to ZZ$ versus $H \to WW$ classification, simply connected and circularly connected graphs fail to capture any meaningful features. This suggests that these graph structures are ineffective when a single particle decays into all final-state objects.

5. Dis-connected and linearly connected graphs are more effective for classifying $H \to ZZ$ events against $H \to WW$ background.

The application of GNNs in HEP is still an emerging area, with limited resources available. A few studies, such as [34] and [5], suggest simply connected graphs for similar analysis, and there is a general tendency to adopt simply connected graphs as a starting point. However, this approach can be ineffective in certain scenarios. Specifically, using `GCNConv` layers with simply connected graphs, where nodes represent particles and their four-momentum are used as node features, often fails to produce any meaningful training outcomes for classification tasks like $H \to ZZ$ versus $H \to WW$.

Impact of node representations

1. The DNN model and `GraphConv`-based GNNs show improvement in performance when using kinematic properties as features (p_{T}, η, ϕ, M) compared to four momentum features (p_x, p_y, p_z, E).

2. However, for `GCNConv`-based GNNs, four momentum coordinates generally yield better results.

Effect of pooling methods

1. Global pooling methods such as Add, Max, and Mean have a minimal impact on overall model performance.

2. Among them, Max pooling provides a slight advantage over Mean and Add pooling, making it the preferred choice.

Comparison of GN layers

1. Generally, `GCNConv` models with a suitable input graph structure show performance comparable to DNNs.

2. `GraphConv` outperforms `GCNConv` and DNNs across most cases, even with fewer trainable parameters.

3. Despite incorporating an attention mechanism, `GATConv` models seem to be outperformed by simpler `GraphConv` models.

4. `MyConv` layer, which dynamically learns edge weights, achieves better performance than `GraphConv`, making it the most optimal choice.

Model choice for event classification

Based on our comparison of different GN layers, `GraphConv` emerges as the most effective layer for event classification, outperforming both `GCNConv` and `GATConv` across various graph structures. However, the performance of these GN layers varies significantly depending on

the input graph structure. `MyConv` layer further enhances classification performance and reduces the sensitivity to the input graph structure. This adaptability makes `MyConv` the most effective layer for event classification. Using the `MyConv` layer with simply connected graphs and max pooling layers is recommended for any generic event classification task.

In addition to event classification, I briefly explored using GNNs for track-based analysis, focusing on identifying events with high transverse momentum (p_{T}) charged particles from tracker hit data.

Various graph representations can be formed from the tracker hits, including point cloud graphs, nearest neighbour graphs, and distance-based graphs. Each representation influenced how the GNN learned spatial relationships and extracted features from the hit data.

The simplest GNN model, with a point set of every hit as a node in the input graph, performed well on single-track and two-track datasets, achieving high AUC values. However, performance declined as the number of tracks increased, highlighting the complexity of distinguishing high-p_{T} tracks in events with higher track multiplicities. More work is needed to develop better models for this classification.

Notably, the ability of GNNs to handle variable input sizes was tested by evaluating models trained on one dataset against datasets with different track multiplicities. This flexibility allowed the model to generalize reasonably across varying levels of event complexity. Reinforcing suitability of GNNs for track-based analysis in HEP.

Overall, this thesis establishes a foundation for graph-based machine learning in collider physics and opens new avenues for future research. The increasing computational capabilities and growing interest in geometric deep learning suggest that GNNs will play an important role in upcoming experimental analyses, ultimately contributing to more precise and efficient methods for understanding experimental data.

Bibliography

[1] M. Tanabashi et al. "Review of Particle Physics". In: *Phys. Rev. D* 98.3 (2018), p. 030001. DOI: 10.1103/PhysRevD.98.030001.

[2] Jon Butterworth. "The Standard Model: How far can it go and how can we tell?" In: *Phil. Trans. Roy. Soc. Lond. A* 374.2075 (2016). Ed. by C. David Garner, p. 20150260. DOI: 10.1098/rsta.2015.0260. arXiv: 1601.02759 [hep-ex].

[3] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. "Searching for Exotic Particles in High-Energy Physics with Deep Learning". In: *Nature Commun.* 5 (2014), p. 4308. DOI: 10.1038/ncomms5308. arXiv: 1402.4735 [hep-ph].

[4] Matthew Feickert and Benjamin Nachman. "A Living Review of Machine Learning for Particle Physics". In: (Feb. 2021). arXiv: 2102.02770 [hep-ph].

[5] Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. "Graph Neural Networks in Particle Physics". In: (July 2020). DOI: 10.1088/2632-2153/abbf9a. arXiv: 2007.13681 [hep-ex].

[6] Jie Zhou et al. "Graph neural networks: A review of methods and applications". In: *AI Open* 1 (2020), pp. 57–81. ISSN: 2666-6510. DOI: https://doi.org/10.1016/j.aiopen.2021.01.001. URL: https://www.sciencedirect.com/science/article/pii/S2666651021000012.

[7] Lyndon Evans and Philip Bryant. "LHC Machine". In: *JINST* 3 (2008), S08001. DOI: 10.1088/1748-0221/3/08/S08001.

[8] CMS Collaboration. "The CMS experiment at the CERN LHC". In: *Journal of Instrumentation* 3.08 (2008), S08004. DOI: 10.1088/1748-0221/3/08/S08004. URL: https://dx.doi.org/10.1088/1748-0221/3/08/S08004.

[9] ATLAS Collaboration. "The ATLAS Experiment at the CERN Large Hadron Collider". In: *Journal of Instrumentation* 3.08 (2008), S08003. DOI: 10.1088/1748-0221/3/08/S08003. URL: https://dx.doi.org/10.1088/1748-0221/3/08/S08003.

[10] ALICE Collaboration. "The ALICE experiment at the CERN LHC". In: *Journal of Instrumentation* 3.08 (2008), S08002. DOI: 10.1088/1748-0221/3/08/S08002. URL: https://dx.doi.org/10.1088/1748-0221/3/08/S08002.

[11] LHCb Collaboration. "The LHCb Detector at the LHC". In: *Journal of Instrumentation* 3.08 (2008), S08005. DOI: 10.1088/1748-0221/3/08/S08005. URL: https://dx.doi.org/10.1088/1748-0221/3/08/S08005.

[12] Tai Sakuma and Thomas McCauley. "Detector and Event Visualization with SketchUp at the CMS Experiment". In: *Journal of Physics: Conference Series* 513.2 (2014), p. 022032. DOI: 10.1088/1742-6596/513/2/022032. URL: https://dx.doi.org/10.1088/1742-6596/513/2/022032.

[13] A.M. Sirunyan et al. "Particle-flow reconstruction and global event description with the CMS detector". In: *Journal of Instrumentation* 12.10 (2017), P10003. DOI: 10.1088/1748-0221/12/10/P10003. URL: https://dx.doi.org/10.1088/1748-0221/12/10/P10003.

[14] Matteo Cacciari, Gavin P. Salam, and Gregory Soyez. "The anti-kt jet clustering algorithm". In: *Journal of High Energy Physics* 2008.04 (2008), p. 063. DOI: 10.1088/1126-6708/2008/04/063. URL: https://dx.doi.org/10.1088/1126-6708/2008/04/063.

[15] J. Alwall et al. "The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations". In: *JHEP* 07 (2014), p. 079. DOI: 10.1007/JHEP07(2014)079. arXiv: 1405.0301 [hep-ph].

[16] Christian Bierlich et al. "A comprehensive guide to the physics and usage of PYTHIA 8.3". In: *SciPost Phys. Codeb.* 2022 (2022), p. 8. DOI: 10.21468/SciPostPhysCodeb.8. arXiv: 2203.11601 [hep-ph].

[17] J. de Favereau et al. "DELPHES 3, A modular framework for fast simulation of a generic collider experiment". In: *JHEP* 02 (2014), p. 057. DOI: 10.1007/JHEP02(2014)057. arXiv: 1307.6346 [hep-ex].

[18] Byron P. Roe et al. "Boosted decision trees, an alternative to artificial neural networks". In: *Nucl. Instrum. Meth. A* 543.2-3 (2005), pp. 577–584. DOI: 10.1016/j.nima.2004.12.018. arXiv: physics/0408124.

[19] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine Learning* 20.3 (1995), pp. 273–297. DOI: 10.1007/BF00994018. URL: https://doi.org/10.1007/BF00994018.

[20] ATLAS Collaboration. "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC". In: *Physics Letters B* 716.1 (2012), pp. 1–29. ISSN: 0370-2693. DOI: https://doi.org/10.1016/j.physletb.2012.08.020. URL: https://www.sciencedirect.com/science/article/pii/S037026931200857X.

[21] CMS Collaboration. "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC". In: *Physics Letters B* 716.1 (2012), pp. 30–61. ISSN: 0370-2693. DOI: https://doi.org/10.1016/j.physletb.2012.08.021. URL: https://www.sciencedirect.com/science/article/pii/S0370269312008581.

[22] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: (2017). arXiv: 1609.04747 [cs.LG]. URL: https://arxiv.org/abs/1609.04747.

[23] Michael M. Bronstein et al. "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges". In: (2021). arXiv: 2104.13478 [cs.LG]. URL: https://arxiv.org/abs/2104.13478.

[24] Savannah Thais et al. "Graph Neural Networks in Particle Physics: Implementations, Innovations, and Challenges". In: *Snowmass 2021*. Mar. 2022. arXiv: 2203.12852 [hep-ex].

[25] Matthias Fey and Jan E. Lenssen. "Fast Graph Representation Learning with PyTorch Geometric". In: *arXiv preprint arXiv:1903.02428* (2019). URL: https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html.

[26] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: (2017). arXiv: 1609.02907 [cs.LG]. URL: https://arxiv.org/abs/1609.02907.

[27] Christopher Morris et al. "Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks". In: (2021). arXiv: 1810.02244 [cs.LG]. URL: https://arxiv.org/abs/1810.02244.

[28] Petar Veličković et al. "Graph Attention Networks". In: (2018). arXiv: 1710.10903 [stat.ML]. URL: https://arxiv.org/abs/1710.10903.

[29] PyTorch Geometric. *Max Pooling Layer*. 2024. URL: https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.pool.max_pool.html.

[30] PyTorch Geometric. *Global Mean Pooling*. 2024. URL: https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.pool.global_mean_pool.html#torch_geometric.nn.pool.global_mean_pool.

[31] PyTorch Geometric. *Global Add Pooling*. 2024. URL: https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.pool.global_add_pool.html#torch_geometric.nn.pool.global_add_pool.

[32] Pushpalatha C. Bhat. "Multivariate Analysis Methods in Particle Physics". In: *Ann. Rev. Nucl. Part. Sci.* 61 (2011), pp. 281–309. DOI: 10.1146/annurev.nucl.012809.104427.

[33] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: (2017). arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[34] Murat Abdughani et al. "Probing stop pair production at the LHC with graph neural networks". In: *JHEP* 08 (2019), p. 055. DOI: 10.1007/JHEP08(2019)055. arXiv: 1807.09088 [hep-ph].