AMERICAN UNIVERSITY OF BEIRUT

# GRAPH NEURAL NETWORK ARCHITECTURES FOR FAST SIMULATION AND MUON MOMENTUM INFERENCE AT THE CMS DETECTOR

by
ALI ASSADALLAH HARIRI

A thesis
submitted in partial fulfillment of the requirements
for the degree of Master of Engineering
to the Department of Mechanical Engineering
of the Maroun Semaan Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
December 2020

AMERICAN UNIVERSITY OF BEIRUT

# GRAPH NEURAL NETWORK ARCHITECTURES FOR FAST SIMULATION AND MUON MOMENTUM INFERENCE AT THE CMS DETECTOR

by

ALI ASSADALLAH HARIRI

**Approved by:**

|  |  |
|---|---|
| Dr. Mariette Awad, Associate Professor<br>Electrical and Computer Engineering | Advisor<br>Mariette Awad |
| Dr. Issam Lakkis, Professor<br>Mechanical Engineering | Chairman and Co-advisor<br>Mariette Awad<br>on Behalf of Dr. Lakkis |
| Dr. Sergei Gleyzer, Professor<br>Department of Physics, University of Alabama | Co-advisor<br>Mariette Awad<br>on Behalf of Dr. Gleyzer |
| Dr. Marwan Darwish, Professor<br>Mechanical Engineering | Member of Committee<br>Marwan |
| Dr. Leen Alawieh, Guest Researcher<br>Department of Mechanical Engineering,University of Rochester | Member of Committee<br>Mariette Awad<br>on Behalf of Dr. Alawieh |

Date of thesis defense: December 10, 2020

# AMERICAN UNIVERSITY OF BEIRUT

## THESIS, DISSERTATION, PROJECT RELEASE FORM

| | | | |
|---|---|---|---|
| Student Name: | Hariri | Ali | Assadallah |
| | Last | First | Middle |

[✓ Master's Thesis] ◯ Master's Project ◯ Doctoral Dissertation

☐ I authorize the American University of Beirut to: (a) reproduce hard or electronic copies of my thesis, dissertation, or project; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

[✓] I authorize the American University of Beirut, to: (a) reproduce hard or electronic copies of it; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes after: **One ___ year from the date of submission of my thesis, dissertation or project.**
**Two ___ years from the date of submission of my thesis , dissertation or project.**
**Three [✓] years from the date of submission of my thesis , dissertation or project.**

| | |
|---|---|
| Ali Hariri | February 3, 2021 |
| Signature | Date |

This form is signed when submitting the thesis, dissertation, or project to the University Libraries

# Acknowledgements

I would like to express my sincere gratitude for being given the opportunity to work on such an exciting topic in collaboration with the CMS experiment at CERN. For that, I would like to thank the following people who have helped make this journey possible:

- Dr. Sergei Gleyzer for allowing me to conduct research on Deep Learning applied to particle physics in addition to his constant supervision throughout this journey to ensure its success. Our discussions and meetings have given me great insights and his mentoring will always be remembered as I tackle new challenges in my career.
- Dr. Mariette Awad, Dr. Issam Lakkis, Dr. Leen Alawieh and Dr. Marwan Darwish who supervised, reviewed my work and continuously provided the needed support to ensure an efficient academic research environment.
- My colleagues Darya Dyachkova, Sitong An, Michael Andrews and Emanuele Usai for the great teamwork and team support that were crucial to my work.
- Christian Hundt, Giuseppe Fiameni and Mozhgan Kabiri Chimeh for their mentoring throughout the 2020 Helmholtz GPU Hackathon.
- Dr. Hazem Hajj, Dr. Zaher Dawy and Martin Gastal for facilitating the logistics of the collaboration and showing continuous support.

On the other hand, it is worth noting that challenges are not limited to academic ones. As I revisit this work in the future, I would like to be reminded of some challenging conditions that hampered the working progress of most graduate students at the time. The period between end of 2019 and late 2020 had been one of the most challenging for Lebanon, while a global pandemic was simultaneously affecting the global community. That being said, I would not have made it through this tough period without the support of my family and friends to whom I dedicate this work.

عَلى قَدْرِ أهْلِ العَزْم تأتي العَزائِمُ

وَتأتي علَى قَدْرِ الكِرامِ المَكارمُ

- المتنبي

# An Abstract of the Thesis of

Ali Assadallah Hariri for Master of Engineering
Major: Mechanical Engineering

Title: Graph Neural Network Architectures For Fast Simulation And Muon Momentum Inference At The CMS Detector

Accurate and fast simulation of particle physics processes is crucial for the high-energy physics community knowing that simulating the particle showers and interactions in the detector is both time consuming and computationally expensive. Classical fast simulation techniques based on non-parametric methods can improve the speed of the full simulation but suffer from lower levels of fidelity. For this reason, alternative methods based on machine learning can provide faster solutions, while maintaining a high level of fidelity. The main goal of a fast simulator is to map the events from the generation level directly to the reconstruction level. In this thesis work, we present novel approaches for Fast Simulation and Muon Momentum inference in High Energy Physics. More specifically, we explore the potential of graph neural networks in various applications including fast simulation of boosted jets and muon momentum estimation in the CMS detector. We introduce a graph neural network-based autoencoder model that provides effective reconstruction of calorimeter deposits using the earth mover distance metric. We also propose to use graph networks to infer the momentum of muons in the Cathode Strip Chambers given their ability to account for the several features affecting the particles' trajectories. We show that graph-based architectures outperform conventional deep learning baselines in terms of accuracy and result in relatively competitive inference and training time. In addition, we optimize our code for training using NVIDIA Nsight Tools and later investigate the scalability of our Fast Simulation graph model on multiple GPUs for which we get speedups of 1.62, 2.19 and 2.73 while scaling from 2 to 4 devices, respectively.

# Contents

# Chapter 1

# Introduction

Early particle accelerators were built in the early $20^{th}$ century to perform in-depth studies of the subatomic properties of particles. Today, multidisciplinary fields ranging from nuclear science to medicine and other industrial applications rely on particle colliders, the largest of which is currently the Large Hadron Collider (LHC) at CERN, Geneva. First operated in 2008, the LHC accelerates beams containing millions of charged particles within a range of milliseconds, colliding them with opposing beams at four main detector points: CMS, ATLAS, LHCb and ALICE. A particle shower emerges within the detectors whose characteristics allow them to amplify the recorded particle-sensor interactions into digital signals to be analyzed by physicists. In-depth analysis of these records is crucial to extract the most relevant information associated with the unraveling of new physics that help explain several phenomena such as the origin of the universe, the properties of its most fundamental forces and the presence of dark matter. The eclectic nature of both the software and hardware components within the LHC makes it a highly complex operational environment, hence the rising need to simulate it. Several simulation software have been developed for deployment in HEP applications. Geant4 [1] is one toolkit type that relies on full simulation techniques accounting for particle-matter interactions and physical boundary conditions specific to the detector environment. On the other hand, Delphes is a simulation framework that uses parametric methods to perform HEP simulations, providing a user-interface and the option to specify several properties such as detector segmentation, etc [2].

Despite their promising results, these simulation techniques require extensive compute resources and are non-universal, meaning that simulation results are specific to one set of detector properties at a time. The emergence of deep learning architectures paved the way for a multitude of applications in HEP including simulation methods. Di Sipio et. al implement a Generative Adversarial Network (GAN) to learn the represenations of jet pairs at the LHC [3]. Buhmann et. al combine Variational Autoencoders and GANs for electromagnetic shower reconstructions at the level of the calorimeters [4].Conventional deep learning architectures perform convolution on grid-like Euclidean data, mainly in the form of images.

In this thesis work, I propose the usage of Graph Neural Networks to learn the representations of jet events resulting from collisions. This methodology allows the treatment of sparse particle hits within detectors as point clouds in 3D space. Therefore, these point clouds could be mapped into a dense graph whose nodes have features indicating the x and y locations of the particle hits and their corresponding measured energy. In addition, the connections between the nodes would be based on the k-nearest neighbour approach in Euclidean space. To proceed, we learn on the jet datasets by means of state of the art graph convolution operations. In [5], Qu and Gouskous show that such a methodology of learning on jet events as graphs is successful as they outperformed all other baselines in the task of jet tagging. In addition, this method allows us to operate on raw datasets containing $\mathcal{O}(100)$ particles instead of conventional $\mathcal{O}(1000)$ particles. That being said, we shed light on the ability of graph-based architectures to learn the properties of collider events' data in latent space for reconstruction purposes, in addition to their computational performance with regards to conventional deep learning techniques. Finally, we investigate the ability of graph networks to generalize to different HEP applications. To proceed, we compare the performance of a GNN model to fully connected networks and convolutional networks for the task of muon momentum inference in the Cathode Strip Chambers at CMS. Such measurements are crucial for improved sample selection within the Trigger system where a momentum threshold is applied and accepts batches of recorded signals accordingly. Having said that, careful muon analysis is crucial to study different physics phenomena, including the Higgs boson decay processes, one of which decays to 4 muons.

In a nutshell, the aim of this thesis work is to discuss the potential provided by graph neural networks (GNNs) in high energy physics applications. Their ability to learn on sparse representations in the form of point clouds while disregarding the blank space surrounding them makes them a suitable candidate for detector data whereas the learning process takes place on the particle hits exclusively. By using them in Fast Simulation and muon momentum inference, we find that GNNs outperform conventional convolution techniques on images in reconstructing and classifying collision events. More importantly, the speedup obtained with respect to traditional Monte Carlo techniques was immense, bearing in mind that even further speedup was noticed during scaling of GNNs on multiple GPUs. In the next chapters, we start by introducing the field of particle physics and the need for particle colliders whose geometry we describe as well. Next, we elaborate on the added value provided by Machine Learning algorithms in this field, and the obstacles tackled with the rise of Deep Learning architectures. Finally, we propose geometric deep learning as an alternative methodology in HEP by using it in two applications: Fast simulation and muon momentum inference. Later chapters discuss the methods and algorithms that we build for our experiments followed by their corresponding results and the metrics adopted for their assessment.

# Chapter 2

# An overview of Particle Physics

In this chapter we discuss the motivation behind particle accelerators, with the LHC currently being the largest one ever built. We shed light on some basic concepts in particle physics related to Standard Model, a theory that describes the building blocks of our universe.

## 2.1 The atom: A short history

Atomic theory dates back to the 5th century B.C in Ancient Greece, where the debate about the origin of matter and its composition sparked controversy among philosophers of the time. Two Greek philosophers, Democritus and Leuccipus, suggested that matter could be divided into smaller homogeneous clusters of one element forming the building block for the universe. This element was named "atomos", an ancient Greek term referring to what was thought to be an "indivisible entity". This theory had been rejected until the 16th century when John Dalton reintroduced the notion of an atom as being specific to each element of the periodic table where it differs in mass and size. Three centuries later, J.J Thomson performed the cathode ray tube experiment where electrodes are connected to a cathode ray which was deflected when placed between two oppositely charged plates [6]. This phenomenon confirmed the presence of a negative charge inside the atom: the electron. Bearing in mind that an atom is neutral in charge, a counter-balancing positive charge should exist. Having said that, Ernest Rutherford performed a gold foil experiment where a thin sheet of gold was hit with a beam of positively charged alpha particles, of which a tiny fraction had been deflected from the gold sheet and observed on a luminescent screen [7]. This observation suggested that the atom is mostly made of empty space containing negatively charged electrons surrounding a tiny positively charged space, later known as the nucleus. Until the mid-twentieth century, scientists considered the proton, electron and the neutron, to be elementary, indivisible particles acting as the building block of the universe. However, technological advances allowing the development state-of-the-art physics equipment have shown otherwise.

## 2.2 The Standard Model

The early twentieth century has witnessed the discovery of cosmic rays. These are high-energy particles mostly composed of protons and charged nuclei traveling at the speed of light in outer space, resulting in high-energy collisions penetrating the Earth's surface. One simple method to study cosmic rays is through cloud chambers where particle tracks could be observed with a flashlight through a black box containing dry ice [8]. As a result, the first subatomic particle was discovered using this equipment: the meson, a particle around 200 times the mass of the electron. At this point, the physics community was highly motivated to launch a hunt for more subatomic particles. As studying cosmic rays from outer space is challenging, one solution is to produce them in the laboratory, which rises the need for devices able to accelerate particles to the speed of light, simulating the environment variables in outer space: we need particle accelerators. By the 1970's, dozens of new particles were introduced, and as years passed, a better understanding was acquired regarding the 4 types of forces recognized today as gravity, electromagnetism, the weak force and the strong force. Today, the Standard Model (SM) is the most accepted scientific theory to decipher and make sense of the "particle zoo" observed throughout the past decades. It explains that matter exists by virtue fermions that include quarks and leptons. Matter is able to interact with the forces of nature by means of bosons. For instance, the photon is the carrier of the electromagnetic force, while the gluon mediates the strong force inside hadrons (protons and neutrons). Finally, the most fundamental element of the SM is the Higgs Boson, named after British theoretical physicist Peter Higgs. To shed light on the importance of this particle sometimes referred to as "God's particle", it is crucial to refer to quantum field theory which considers that the universe is made of a quantum fields whose excitation results in a Higgs particle that would give protons, electrons and other particles, their known mass: this is the "Higgs mechanism".

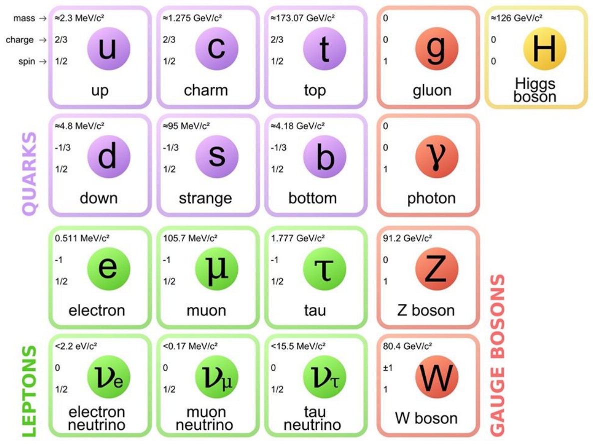


Figure 2.1: The Standard Model [9]

## 2.3 The Large Hadron Collider

Early twentieth century has seen the operation of the first particle accelerators starting from UC Berekeley's cyclotrons in the 1930's (awarded Nobel Prize in 1939) to CERN's circular Proton Synchotron (PS) in 1959, ending up with Stanford's Linear Accelerator (SLAC) in 1966. Today, the world's largest particle accelerator is the Large Hadron Collider (LHC) which operates at the European Organization for Nuclear Research (CERN) in Geneva.

In contrast to earlier linear accelerators, the LHC is a circular accelerator with the following characteristics: 32 km in diameter, 27 km in circumference and an underground depth between 50 and 175 meters [10]. The overall structure crosses the Swiss-French borders. It is made of superconducting magnets operating at -271.3°C, accelerating beams of either protons or lead nuclei in opposite directions at the speed of light, and colliding them at four main collision points to be detected by four detectors: CMS, ATLAS, LHCb and ALICE [11]. The outstanding physical properties of the LHC make it a suitable candidate to unravel new discoveries through the production and the monitoring of rare physics signatures at high luminosity collisions of proton or heavy ion beams. For instance, proton beams in the LHC are introduced with an injection energy of 450 GeV, travelling in vacuumed beams at 99.9999% the speed of light, giving an eventual total collision energy of 14 TeV resulting from two opposing beams holding proton energies of 7 TeV each [12][13]. Collision events must occur in a timely manner to pave room for detectors to perform fast data pre-processing to reject irrelevant signals (See Chapter 4). Having said that, the LHC is characterized by a bunch crossing frequency of 40 MHz, meaning that two consecutive proton/heavy ion bunches are separated by 25 ns. Finally, one of the main aspects of a particle accelerator is the luminosity, a metric that indicates the number of collisions per $m^2$ per second. As a matter of fact, future LHC operations are planned to run at luminosity increased by a factor of 10 [14].

The LHC started operating in 2008 in the aim of unraveling new physics, which was achieved 4 years later in July 2012 by the CMS and ATLAS collaborations. The latter observed a signal at an invariant mass of 125 GeV from diphoton and 4 lepton decay channels. Their data was compatible with the Standard Model's Higgs boson theoretical description, and hence the two collaborations had announced the long awaited discovery of the Higgs boson. This event has paved the way for further research to be done in particle physics, especially for further analysis of the Higgs boson behavior. In fact, recent CMS experiments succeeded in the measurement of Higgs boson decay rates, productions and Yukawa coupling to fermions on the one hand [15], and the observation of the Higgs decay to two bottom quarks on the other hand [16].

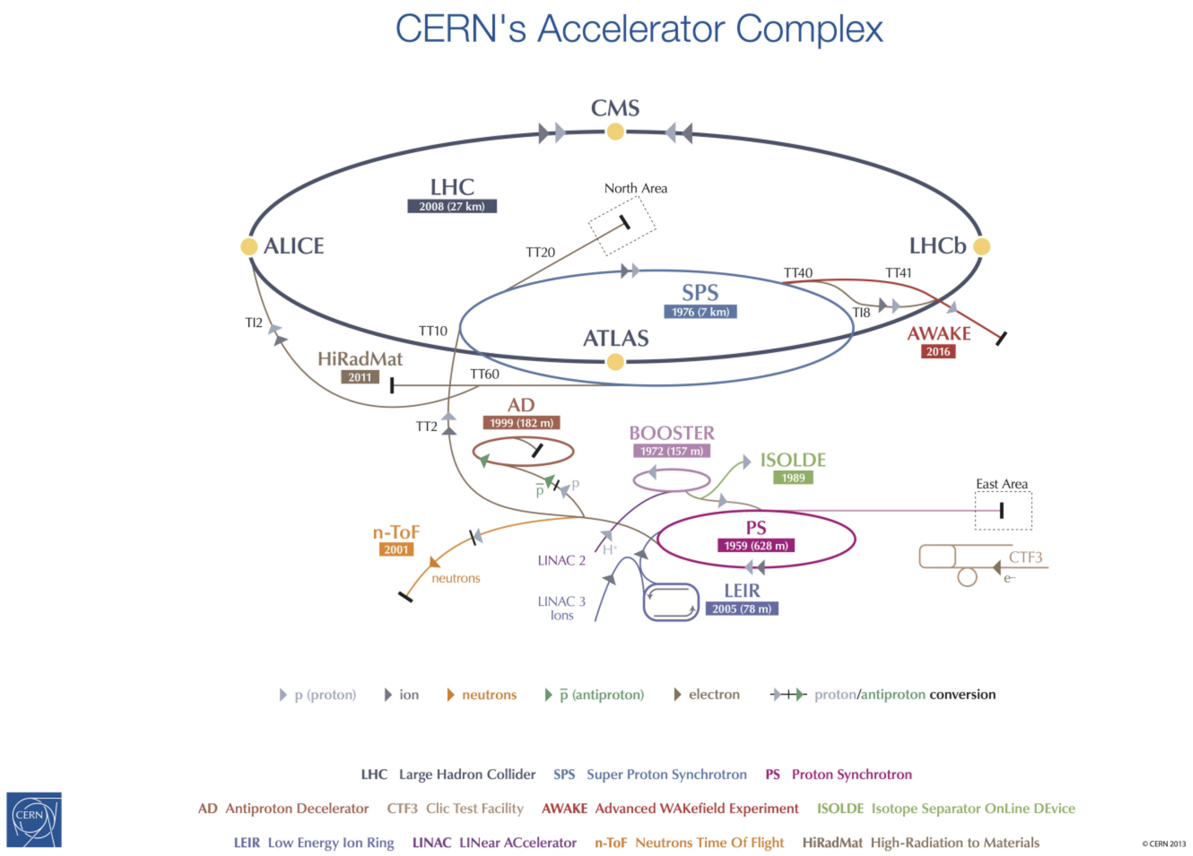


Figure 2.2: Full diagram of the LHC operating detectors and accelerators [17]

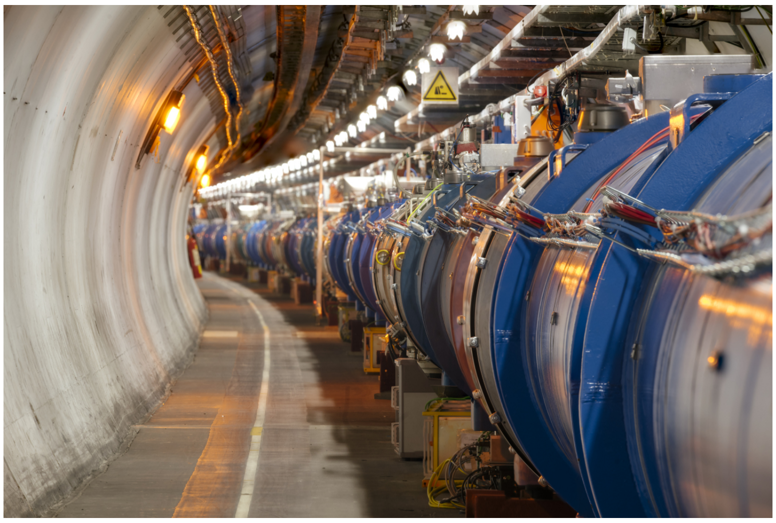


Figure 2.3: The LHC 100 meters underground [18]

## 2.4 What is a Jet?

In particle physics, jets are most commonly known as experimental signatures spotted in the detector deriving from the hadronization of quarks and gluons as a result of high energy collisions taking place in particle colliders. As a matter of fact, quarks and gluons cannot exist in a free state in nature due to the colour confinement suggested by Quantum Chromodynamics (QCD). Instead, quarks and gluons holding fragments of coloured charge will decay into further coloured fragments that eventually combine into colourless objects hence satisfying the colour confinement rule. The process undergone by the quarks and gluons is called hadronization, a process that results in a shower of hadrons measured at the level of the different detector layers: the tracker systems, electromagnetic calorimeters and hadronic calorimeters. The resulting shower of hadrons forms a narrow cone-like jet whose components are crucial for physicists to trace the origin of this decay and associate it with a type/flavour of quarks out of 6: up, down,strange,charm,bottom and top. This is done by means of jet reconstruction algorithms whose details are out of the scope of this thesis.

# Chapter 3

# The CMS Detector

The Compact Muon Solenoid is a multi-purpose detector used to study particle properties following beam collisions. As the name suggests, this machine consists of tightly packed materials that create a suitable operational environment for physics experiments. In this section, we give an overview of the CMS detector geometry, the different sub-layers that compose it, and the outer layers serving as muon detectors.

## 3.1 Structure

The CMS detector is 15 meters high and 21 meters long [19]. One main component is the superconducting solenoid magnet which consists of cylindrical coils providing a magnetic field of 4T under operation, which equals 100, 000 times Earth's magnetic field ! As will be discussed later, this magnetic field causes bending of the post-collision particles, allowing more precise measurements to take place. The "onion-shaped" structure of the CMS detector encloses a set of overlapping sub-detector layers going outwards from the collision point and allowing the extraction of crucial information about post-collision particles including their energy, momenta, and trajectories.

## 3.2 The Silicon Tracker system

The innermost sub-detector is a tracker system composed of almost 16,000 silicon strip modules connected to readout electronics. Silicon sensors have a high granularity, thus providing a good spatial resolution and response to particle hits. The latter travel through the silicon strips, from which an avalanche of electrons escapes in response to the charge of the colliding particles, resulting in a readable pulse signal with a duration of a few nanoseconds. This signal is made readable by APV25 chips which amplify it and provide us with the hit locations allowing the path reconstruction process to take place [22].

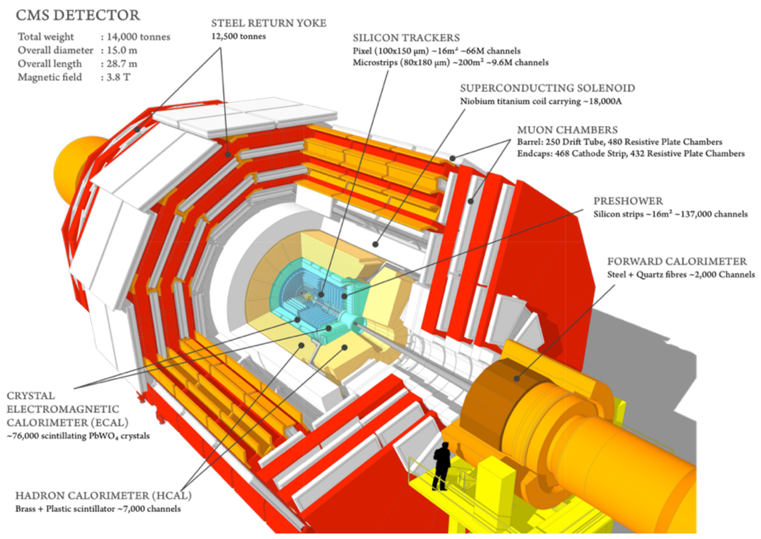

CMS DETECTOR
Total weight : 14,000 tonnes
Overall diameter : 15.0 m
Overall length : 28.7 m
Magnetic field : 3.8 T
STEEL RETURN YOKE
12,500 tonnes
SILICON TRACKERS
Pixel (100x150 μm) ~16m² ~66M channels
Microstrips (80x180 μm) ~200m² ~9.6M channels
SUPERCONDUCTING SOLENOID
Niobium titanium coil carrying ~18,000A
MUON CHAMBERS
Barrel: 250 Drift Tube, 480 Resistive Plate Chambers
Endcaps: 468 Cathode Strip, 432 Resistive Plate Chambers
PRESHOWER
Silicon strips ~16m² ~137,000 channels
FORWARD CALORIMETER
Steel + Quartz fibres ~2,000 Channels
CRYSTAL ELECTROMAGNETIC CALORIMETER (ECAL)
~76,000 scintillating PbWO₄ crystals
HADRON CALORIMETER (HCAL)
Brass + Plastic scintillator ~7,000 channels

Figure 3.1: Layers and components of the CMS detector [20]

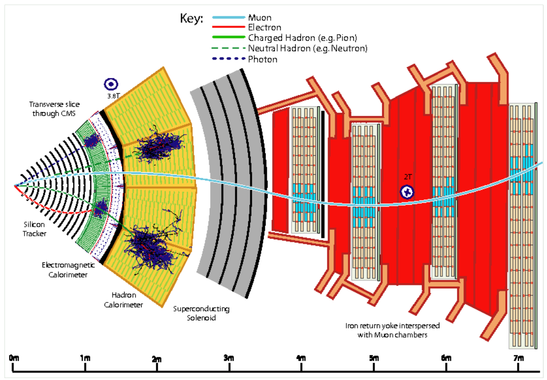

Key:
Muon
Electron
Charged Hadron (e.g. Pion)
Neutral Hadron (e.g. Neutron)
Photon
3.8T
Transverse slice through CMS
Silicon Tracker
Electromagnetic Calorimeter
Hadron Calorimeter
Superconducting Solenoid
2T
Iron return yoke interspersed with Muon chambers
0m
1m
2m
3m
4m
5m
6m
7m

Figure 3.2: Section view of the CMS detector showing different particle trajectories [21]

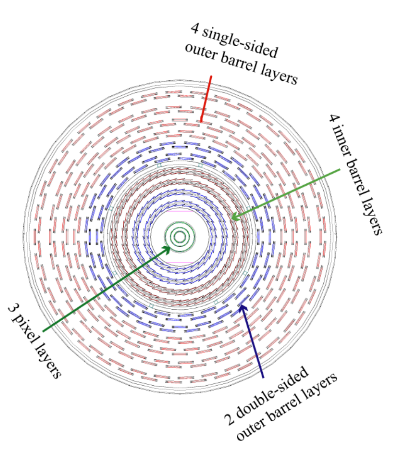

4 single-sided outer barrel layers
4 inner barrel layers
3 pixel layers
2 double-sided outer barrel layers

Figure 3.3: Silicon tracker strips [23]

## 3.3 Calorimeters

The second and third innermost sub-detector layers are the Calorimeters. In contrast to the tracker system, whose purpose is to identify hits for track reconstruction, a calorimeter measures a particle's energy. That being said, the CMS consists of two such detectors: The Electromagnetic Calorimeter (ECAL) and the Hadronic Calorimeter (HCAL).

As the name suggests, the ECAL measures energy deposits from electromagnetic particles; i.e photons and electrons. This is achieved by the means of the lead tungstate ($PbWO_4$) crystal sensors whose physical properties such as a short radiation length $X_0 = 0.89\ cm$ and small Moliere radius of $R_M = 2.2\ cm$ make them suitable materials to interact with electromagnetic particles [19]. Furthermore, TableX in Appendix B shows a comparison between several physical properties of ($PbWO_4$) and another set of materials for potential detector usage. It is observed that, among the samples, lead tungstate possesses the highest material density ($8.28\ g/cm^3$), the highest refraction index (2.30) and the lowest light decay time in nanoseconds [24]. Having said that, it is clear that $PbWO_4$ presents the better characteristics of a scintillator material to be placed in a high irraditation facility such as the LHC. This is due to this material's ability to withstand high magnetic fields and radiations in addition its fast photon burst given its relatively low light decay time. In total, the CMS detector consists of around 76 000 crystals of $2.2 \times 2.2\ cm$ front shape and a length of 23 $cm$ [25]. These densely packed crystal towers are distributed as follows: around 61000 towers in the central barrel section (EB) with a pseudo-rapidity coverage range up to $|\eta| = 1.48$ and 14600 towers in the 2 surrounding endcap sections (EE) with $1.479 < |\eta| < 3.0$ [24]. That being said, ECAL's crystal density property allows better measurements of particle hit energies at higher resolutions due to reduced fluctuations. The energy resolution at the ECAL's EB section is measured as follows:

$$\frac{\sigma}{E} = \frac{2.8\%}{\sqrt{E}} \oplus \frac{12\%}{\sqrt{E}} \oplus 0.3\% \tag{3.1}$$

where the first two terms of the equation correspond to the stochastic and noise contribution to the resolution, respectively while the third term is a constant and E has units of GeV [26].Attached to these crystals are photodetectors that sense the bursting light and convert it to an electrical signal.

On the other hand, the Hadronic Calorimeter (HCAL) measures energy deposits from charged and neutral hadrons, the most known of which are protons and neutrons. Similar to the ECAL, the HCAL's total of 9072 towers are distributed between different sections: barrel (2592), endcap (2592), outer (2160) and forward (1728) [27]. Nevertheless, the grid topology of the HCAL is much sparser than the ECAL's due to different segmentations; ECAL has a fine-grained mesh in

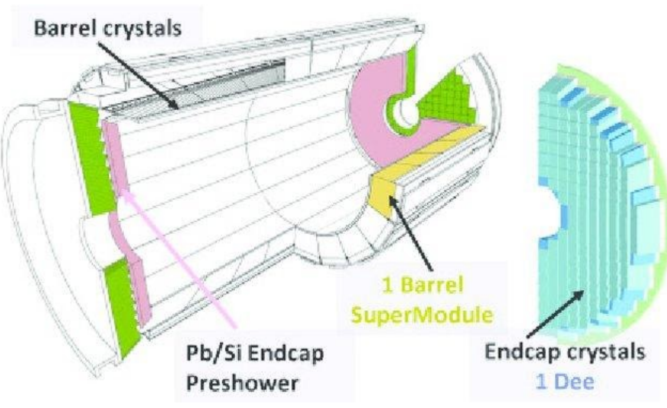


Figure 3.4: Schematic view of the CMS Electromagnetic Calorimeter [24]

contrast to HCAL coarser granularity. In terms of material selection, the HCAL is mainly composed of alternating layers of brass plates and plastic scintillators. Having said that, hadrons interact with layers of fluorescent materials, resulting in a light pulse following a particle hit. At this stage, precise measurements of a hadron's energy and position are made possible through specially designed optic fibers that read the light signal at a given hit location and send it to be amplified and read out. The combined resolution given by CMS' ECAL+HCAL is given by:

$$\frac{\sigma}{E} = \frac{110.7\%}{\sqrt{E}} \oplus 7.3\% \tag{3.2}$$

## 3.4 Muon Chambers

As the name "Compact Muon Solenoid" indicates, CMS is made of compact concentric cylindrical detector layers of heterogeneous composition. One of the main characteristics of this machine is its ability to bend high-momentum muons, detect them and measure their corresponding momenta. As mentioned in Section 1, muons have a mass 200 times bigger than that of the electron. Hence, these high energy particles cannot be stopped by the calorimeters and will be detected outwards in the muon chambers. For this purpose, a magnetic field of almost 4T resulting from a highly conductive solenoid magnet is crucial to monitor particles as they move away from the Leading Vertex (LV). On the one hand, it allows the identification of the particle charge, as opposite charges move in opposite directions. On the other hand, for a known magnetic field value, a particle's momentum can be estimated from the bending curvature in its trajectory; high-momentum particles present lower curvature along their path. In a nutshell, the magnetic field bends the muon trajectory allowing the calculation of its momentum.

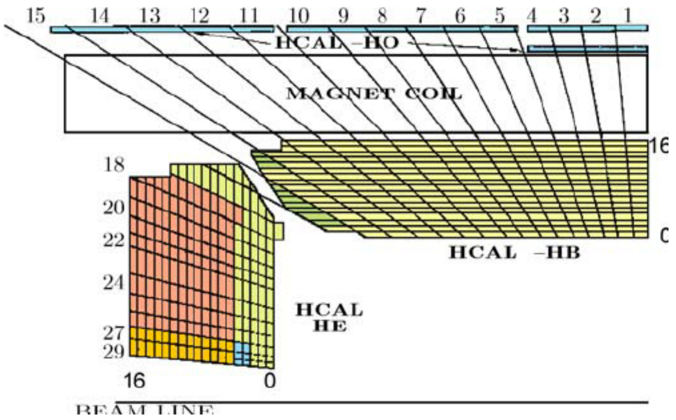


Figure 3.5: HCAL segmentation [28]

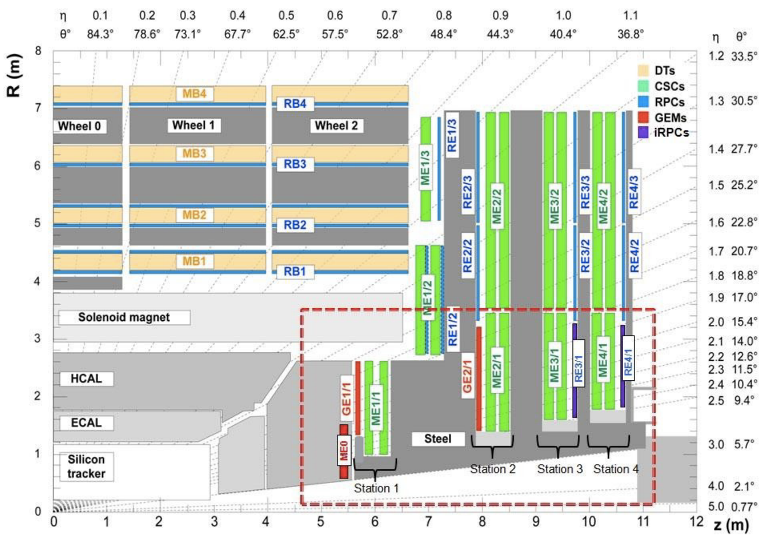


Figure 3.6: Section view of the ME chambers composing the muon detectors [29]

# Chapter 4

# Big Data and LHC Physics

Throughout history, several particle physics breakthroughs were achieved with hardware-based setup such as bubble chambers and other detector technologies, taking the example of the previously mentioned experiments performed by J.J. Thomson followed by Ernest Rutherford, two discoveries that shaped our understanding of the atom. Nevertheless, the LHC is one of the biggest experimental setups ever built and records events at a rate that is impossible for a human inspection to take place. Several detectors spread around the LHC contain well-calibrated hardware components reading billions of events and sending 30 petabytes worth of data to the control center every year. Due to limited resources, state-of-the-art algorithms are continuously developed at CERN to purely select events with a high potential of uncovering new physics, while rejecting noise. For this purpose, CMS contains a trigger system that operates in several steps to accept or reject a post-collision event.

## 4.1 Level 1 Trigger

The Level 1 trigger is a hardware system with a fixed latency. As mentioned previously, the ECAL, HCAL and Muon Chambers detect different types of particles. Therefore, these layers of CMS undergo separate trigger procedures. On the Calorimeter level, energy measurements from all hits are first sent to a regional calorimeter trigger (RCT) where e/y candidate hits are sampled and sent to the Global Calorimeter Trigger. The latter uses the total energy deposits' sums ($E_t$) and clusters jets. On the other hand, data from the CSC and DT muon chambers is first analyzed at the level of each of their corresponding stations individually (ME 1/1, etc). At this stage, front-end electronics collect hit information throughout each given station and send them through optical fibers to the regional track finders. The latter select the suitable muon candidates and estimate their momenta by measuring the curvature along their trajectory with a known magnetic field. Finally, the accepted batch of potential muon candi-

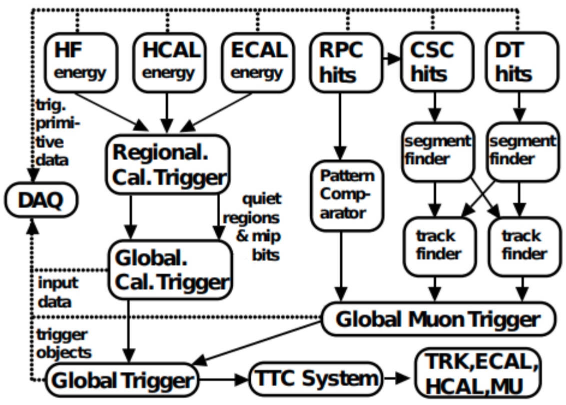


Figure 4.1: Flow chart summarizing the trigger system operations [30]

dates is transferred to the Global Muon Trigger (GMT), where further sampling is needed to ensure that no muon is read by multiple trigger systems [30].

## 4.2 Machine Learning in High Energy Physics

Following the recent breakthroughs in computing hardware, software tools are able to simulate highly complex and stochastic processes in several disciplines. In particle physics, Monte Carlo simulations are used as event generators that follow a pre-defined set of parameters and are later compared to real detector data where significant statistical conformity with theoretical models could be an indicator of new physics. Currently, several fast-simulation tools such as Geant4, Pythia and other state-of-the-art reconstruction software are being developed at CERN, providing a computationally expensive, yet highly parallelizable frameworks that notably alleviate the cost from real experiments. At later stages, simulated and real events need to be scrutinized and sampled. This rises the need to a data analysis scheme that is able to find patterns and extract high and low-level features from data collected by events previously described as complex and stochastic in nature. In this chapter we shed light on the contributions of Machine Learning algorithms to high-energy physics by shedding light on a set of applications where those methods provided in state-of-the-art results, outperforming all traditional algorithms that account for physics knowledge.

Machine Learning tools are nowadays indispensable in the analysis of post-collision events. Their applications are mostly concentrated within the hardware components of the trigger systems, making crucial decisions on what sample of events to consider for further analysis which could include several tasks. For instance, in [31], the CMS collaboration used a Boosted Decision Tree to infer muon momentum from the Level-1 trigger in the muon endcap chamber. They were able to reach a 95 % efficiency in distinguishing muons with a momentum above the threshold value of 25 GeV. Other BDT applications include the study of the geometrical features of tracks resulting from hadronic decays for heavy-quark jet and muon identification [32][33]. In addition, machine learning was crucial in the Higgs Boson discovery and the confirmation of the standard model's predictions regarding the Higgs decays. BDTs were trained to recognize diphoton signals resulting from decay processes and isolate artificial photon signals. The accepted batch is then split into clusters whose mass distributions could indicate the presence of a Higgs signal [34]. ATLAS collaboration used BDTs to study the decay of the Higgs Boson into tau leptons $Z \to \tau^+\tau^-$. Data samples from the LHC Run 1 were split into 6 kinematic regions, each of which was trained on a BDT. Combined results showed evidence of Higgs Boson coupling to tau lepton pairs [35]. CMS and LHCb observed the decay of the strange $B_s$ meson into muon-antimuon pairs ($B \to \mu^+\mu^-$), an event having a probability of 3 in a billion. Having said that, such measurements are very sensitive to noise and require careful data analysis for such a signal to be observed. Analysis was done on p-p collision data at 7 TeV and 8 TeV collected in 2011 and 2012, respectively. Following data preprocessing where event selection takes place, BDTs were trained on a multitude of variables such as to separate signal-events from background events. Such variables include particle momenta estimations, track qualities, etc. Experimental results confirmed the theoretical ones corresponding to the standard model behavior with standard deviations of 3 and 6, respectively [34].

In a nutshell, machine learning techniques require physicists to perform feature engineering on the collected data in order to use them for further analysis. Field-specific algorithms are used to extract relevant information from raw collision data to perform primary tasks such as jet clustering and particle track reconstruction. These tasks pave the way for higher-level information such as particle momenta, hit energies and jet tags. Despite yielding notable improvements with regards to traditional algorithms used in particle physics, machine learning techniques result in high information losses. In [34], a wide range of kinematic information was left unused. In addition, the need for feature-engineering makes machine learning techniques are considerably computationally expensive. As CERN aim to continuously increase luminosity through scheduled updates [14], higher amounts of data are expected for analysis, hence the massive need for novel algorithms for analysis which could explore more features with decreasing human intervention.

# Chapter 5

# Deep Learning

Continuous efforts have been made at CERN for the development of algorithms efficient enough to accommodate the large data volumes while doing fast analysis. The rise of deep neural network architectures and the breakthrough in the GPU hardware technologies paved the way for numerous applications in multidisciplinary fields that rely on the current computer vision algorithms. The latter are at the core of novel techniques used in molecular research [36][37] and video tracking [38]. In this chapter, we introduce some core concepts in deep learning and what makes those algorithms successful in learning on large amounts of data.

## 5.1 The human brain motivation

The complex anatomy of the human brain has been scrutinized by the scientific community for a notable period of time. In fact, early theories about the brain date back to the 4th century B.C when Greek philosopher Hippocrates considered it to be "the seat of intelligence", in contrast to Aristotle's suggestion that the brain is merely a cooling mechanism for the blood [39]. The invention of the first microscopes in the 19th century was crucial for neuroanatomists of the time to offer better insights on the functioning of the brain. In 1863, Otto Dieters described the first model of what we currently know as the neuron, a type of cells composing the nervous tissue and characterized by a set of multi-sized branching known today as dendrites and axons [40]. An inter-connected network of such neurons is at the center of the nervous system in the human brain. The discovery of the synapses in 1932 was crucial to explain the communication taking place between these neuron cells by means of chemical and electrical signalling. A signal is propagated throughout one neuron's axon whose terminal contains multiple branches forming a network of connections with thousands of post-synaptic cells (receiving neurons). At this stage, the firing of an action potential causes the release of chemical neuro-transmitters that propagate towards the post-synaptic cells of target neurons. The latter collect incoming signals from all their adjacent

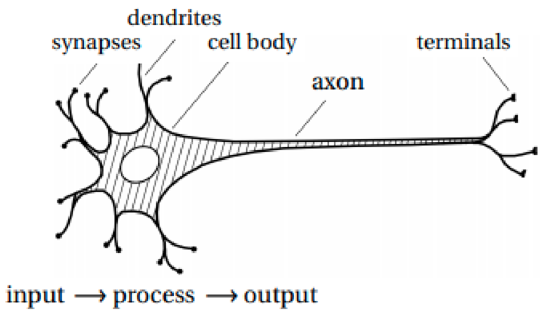


Figure 5.1: Strcuture of one biological neuron [41]

pre-synaptic cells (sending neurons). In case the overall combination overcomes a certain chemical threshold known as the action potential, a resulting signal is induced and propagates further throughout this target neuron's body towards the next sequence of post-synaptic cells (Figure 5.1).

The above description forms a simplified yet fair explanation of the biological neural networks considering the scope of this thesis.

## 5.2 The neuron: A mathematical insight

As mentioned in the previous sub-section, we aim to derive a mathematical formulation describing biological networks in the aim of developing a functioning neural network to be used in computer programs. Having said that, we reiterate that a neural network is a branching between individual neurons. In mathematical terms, these neurons operate using two main vectors: the weight vector w and the bias vector b. Both vectors w and b are referred to as "model parameters" given that they are a property of a larger model architecture. In addition, these vectors are "trainable", meaning that the entries' values will change over several iterations as part of an optimization process. Let $X = [x_1, x_2, x_3, \ldots, x_n]$ and $W = [w_1, w_2, w_3, \ldots, w_n]$ be the input feature vector and the weight vector, respectively. An individual neuron within a specific layer operates on the input as follows: $z = w^T.x + b$ where b is the bias vector.

Having shed light on the operation of a single neuron, we now define the formulation for a full layer of neurons. As shown in Figure 5.4, the weights resulting from a previous layer l-1 are propagated towards all the neurons of the following layer l. We refer to this type of neural networks as "Fully-Connected Networks", given that full connections exist between any neuron of one layer and all the neurons of the following layer. It is worth noting that the numbers of neurons composing a specific layer can be unique i.e if layer l-1 has $n^{[l-1]} = 64$ neurons, the number of neurons $n^{[l]}$ in layer l can be smaller (32), larger (128) or equal to $n^{[l-1]}$.

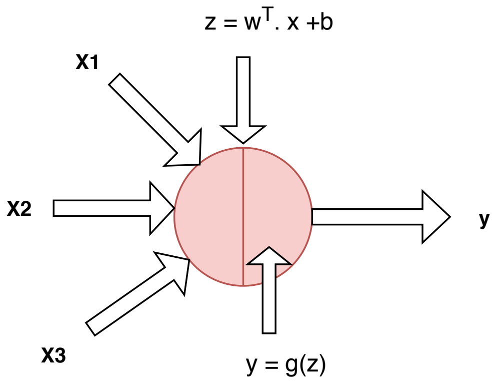


Figure 5.2: Structure of a single neuron wihtin a network

## 5.3 The Activation Function

The activation functions perform a set of mathematical operations on the input data and introduce non-linearities to the network, thus making it possible to learn a wide range of complex tasks. Without an activation function, the output of the perceptron would be a simple linear function, easy to solve, but very redundant against complex topologies.There are many activation functions that could be used with the aforementioned features: the most commonly of which being the ReLU activation, which we will use in this thesis work.

### ReLU

The ReLU (Recti and Linear Unit) has become the most used activation function in recent years and is given as follows:

$$f(x) = max(0, x) \tag{5.1}$$

The behavior of this function, as shown in Figure 5.3, consists in setting the output to 0 for x less than 0 and to exactly replicate the input to the output, if x greater than 0.

### Sigmoid

This Sigmoid function is an S-shaped function bounded between 0 and 1 on the vertical axis and given by as:

$$S(x) = \frac{1}{1 + e^x} \tag{5.2}$$

This function is most commonly used in the output for binary classification problems: Given two classes 0 and 1, the entries with sigmoid output ¿0.5 belong to class 1, otherwise belonging to class 0.

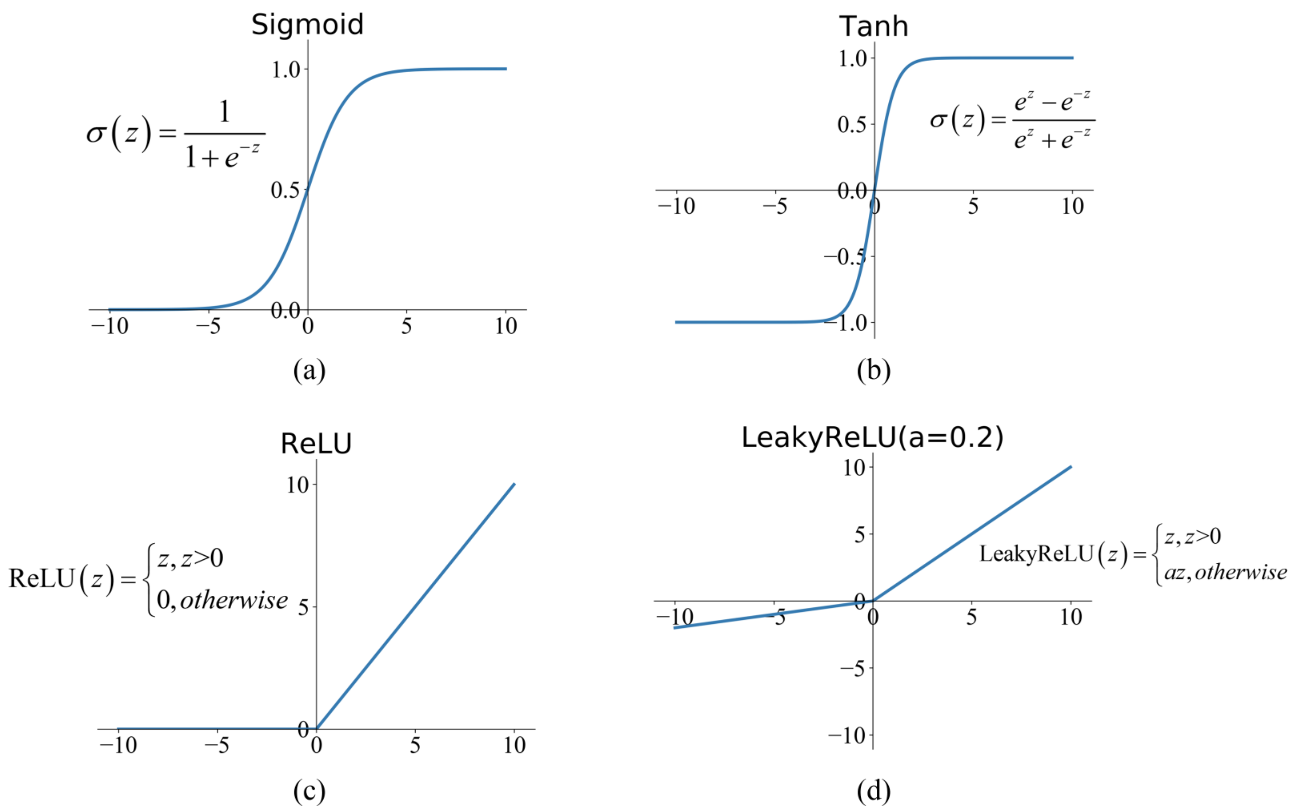


Figure 5.3: A list of the most commonly used activation functions in neural networks

**Hyperbolic Tangent (Tanh)**

The hyperbolic tangent is sigmoidal in shape (S-shaped) but has a range of [-1,1] instead. This allows such an activation function to operate on negative values as well as positive ones.

## 5.4 Fully Connected Networks

The previous sub-section describes the operation of a single neuron. At this point, we shed light on the operation of entire layers of neurons within a neural network. As mentioned previously, each layer l-1 containing $n^{[l-1]}$ neurons will propagate the weights to a following layer l containing $n^{[l]}$ neurons. To proceed, vectorization of each neuron's weights is needed to prevent working with for loops, which makes it computationally inefficient to train. In other words, weight vectors of layer l are stacked vertically into one vector W of shape $n^{[l]} \times n^{[l-1]}$. As a result, the hidden representation of the data at layer [l] is obtained using

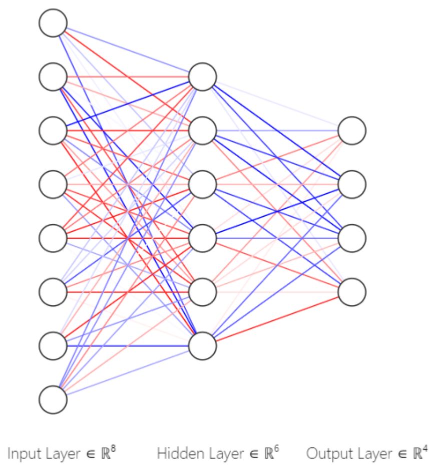


Figure 5.4: A fully-connected network showing with 3 hidden layers

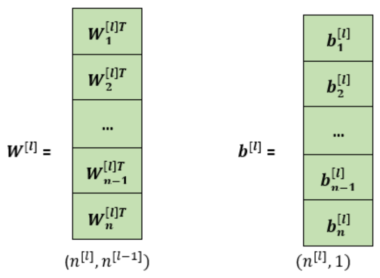


Figure 5.5: Weights and biases

the following equation:

$$z^{[l]} = W^{[l]}.a^{[l-1]} + b^{[l]} \tag{5.3}$$

Next, an activation function is applied to the output of hidden layer [l] in order to introduce non-linearity, hence the input to layer [l+1] is given by:

$$a^{[l]} = g^{[l]} z^{[l]} \tag{5.4}$$

As the name indicates, Fully-Connected Networks (FCNs) are characterized by full connections between the neurons of subsequent layers, i.e the neurons of one layer are fully connected to the activations of the previous layer on the one hand, and their activated signal is shared with the entire networks of neurons of the next layer. Given an input of any type, the FCN acts as a universal function approximator characterized by a multi-layer perceptron aiming to associate each instance of this input with a class (Classification task) or a value (Regression task).

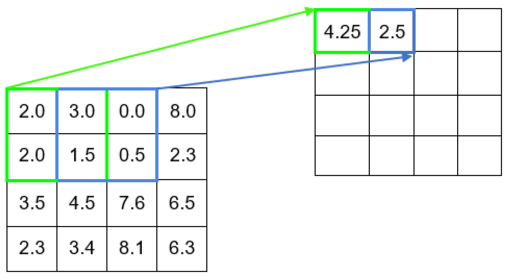


Figure 5.6: Convolution operation with stride 1

## 5.5 Convolutional Neural Networks

A convolutional neural network is a Deep Learning algorithm having its root in Computer Vision originally for the purpose of image processing [42]. This technique owes its success in capturing image features to convolutional kernels. As shown in Figure 5.6 these kernels are best described as windows of fixed size (in this case 2x2) sliding across a subset of pixels in an incremental process until the entire image span is covered. These filters are characterized by a set of randomly initialized yet trainable parameters/weights that are used to convolve the image pixels. The amount of horizontal and vertical kernel displacements across the span of an image is defined by the stride. For instance, a stride of 1 corresponds to an increment of one pixel horizontally until the entire width is convolved by the kernel filter. At this point, the kernel filter slides back to the first column with an increment of 1 pixel downwards and performs the same operation. That being said, the stride, the kernel size in addition to the number of filters to be used are manually defined. In fact, the higher the complexity of the data features and dependencies, the more kernel filters are needed to process such complexities.

## 5.6 ConvNets: A Neuron perspective

In contrast to Fully-Connected Networks, ConvNets have sparser connections within the neurons. The latter are defined in a three-dimensional space by the network's width, height and depth. The width and height parameters are equivalent to the size of the kernel filter convolving the input image. Within this part of the network, the connectivity is local in nature, i.e each neuron processes the set of pixels/entries of size width x height associated with its corresponding region of the image. For instance, Fig. 5.7 below shows that the resulting output neurons are only connected to the previous layer's neurons if the latter were part of the kernel, i.e their receptive field. On the other hand, the depth to the network corresponds to the number of channels present in the input. Neurons are fully connected along this region.

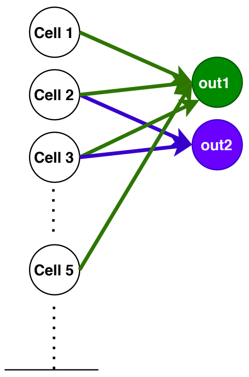


Figure 5.7: Corresponding neural connection to the convolutions in Fig 5.6

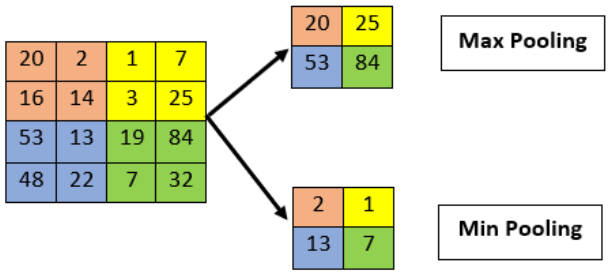


Figure 5.8: Max and Min Pooling examples

## 5.7 Pooling

Real world datasets, mainly 3D images and videos, are often dense. They are characterized by a multitude of channels and considerably high resolution. As seen in the previous section, convolutional kernels preserve an input image's size provided that a stride of 1 is chosen. That being said, pooling layers are introduced to further reduce the dimensionality of the input and alleviate the computational load required to process the data throughout the training process [43]. We distinguish between three main types of pooling: Max Pooling (most often used), Average Pooling and Min Pooling. Depending on the choice, pooling layers slide across the input data and return the maximum, minimum or average pixel value present within a window frame whose size is manually set (Figure 5.8). In addition to dimensionality reduction, pooling layers are helpful in noise cancellation on the one hand, and the extraction of the most representative features on the other hand. In other words, pooling operations help us retain the most important semantics in an image.

## 5.8 Mathematical details

So far we have discussed neural networks in terms of interconnected sets of neurons that propagate input signals across them, perform convolutional and pooling

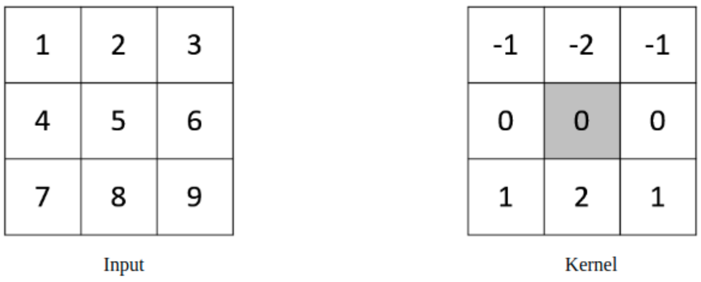


Figure 5.9: Convolution kernel

operations on multiple-channels by means of filters sliding by a pre-defined stride, pass these signals through FCNs and accordingly determine the class/value of each datapoint in this input. While this description provides a good visual insight into the entire signal propagation process, it does not cover the underlying mathematics behind it. Given an input image f, a convolutional kernel h is applied to this image by operating on its pixel values as follows:

$$G[m,n] = (f * h)[m,n] = \sum_j \sum_k h[i,j] f[m-i, n-j] \tag{5.5}$$

where m and n are the row and column indices of the resulting matrix. The convolution operation consists of multiplying each element of the kernel by the corresponding element location in the image. Eventually, a summation of the products is performed, and the resulting integer is the corresponding entry of the output matrix. For square kernels, we consider the kernel center pixel to have a coordinate of [0,0] and label the neighbouring pixels accordingly. To better explain this process, an example is provided in Fig 5.9

The output matrix' entry at location [1,1] is given by:
$y[1,1] = \sum_j \sum_i x[i,j] \cdot h[1-i, 1-j] = x[0,0] \cdot h[1,1] + x[1,0] \cdot h[0,1] + x[2,0] \cdot h[-1,1] + x[0,1] \cdot h[1,0] + x[1,1] \cdot h[0,0] + x[2,1] \cdot h[-1,0] + x[0,2] \cdot h[1,-1] + x[1,2] \cdot h[0,-1] + x[2,2] \cdot h[-1,-1] = 1 \cdot 1 + 2 \cdot 2 + 3 \cdot 1 + 4 \cdot 0 + 5 \cdot 0 + 6 \cdot 0 + 7 \cdot (-1) + 8 \cdot (-2) + 9 \cdot (-1) = -24$

## 5.9 Stride and Padding

In the previous examples we assumed that the kernel filters slide across an image at a rate of one pixel both horizontally and vertically, resulting in a reduced feature map of size (n-w) x (n-w) where n and w are the sizes of the input image and kernel filter (both square for simplicity), respectively. Yet, we have the flexibility to specify the size of the resulting feature map by means of two additional parameters: stride and padding. In fact, the horizontal and vertical pixel displacement of the filter window could be manually adjusted. For instance, a stride of [2,2] will operate on an image by skipping two pixels from the left horizontally following a convolution. Having covered the entire horizontal scope, the filter displaces vertically downwards by 2 pixels and proceeds. Having said that, the stride parameter clearly affects the output shape of the feature map. On the other hand, the padding parameter considers whether the feature map preserves the same output shape as the input image, in which case the type of padding is called 'SAME' and is characterized by adding zero rows and columns to overcome the overlapping issue of the window filter. Otherwise, the feature map is reduced under the 'VALID' padding type which terminates once the kernel filter has covered the entire image scope.

$$Output\ Width = \frac{W - F_w + 2P}{S_w} + 1 \tag{5.6}$$

$$Output\ Height = \frac{H - F_h + 2P}{S_h} + 1 \tag{5.7}$$

**Same Padding**

$$Output\ Width = ceil(\frac{W}{S_w}) \tag{5.8}$$

$$Output\ Height = ceil(\frac{H}{S_h}) \tag{5.9}$$

**Valid Padding**

$$Output\ Width = ceil(\frac{W - F_w + 1}{S_w}) \tag{5.10}$$

$$Output\ Height = ceil(\frac{H - F_h + 1}{S_h}) \tag{5.11}$$

## 5.10 The Loss Function

In the previous sections we covered the forward propagation process for neural networks. We shed light on convolution operations used to detect features within images, the signal propagation between neurons of convolutional layers and fully connected layers, and the latter's ability to provide a classification or regression score. At this point, we define the backpropagation process which is the key element for training neural networks by means of iterative techniques. To reiterate, neural networks are better considered as very complex function approximators, whose parameters we want to optimize. Prior to explaining the optimization process for a neural network, we start by introducing it through a basic example. Figure 5.10 shows the plot of a function $f(x) = 0.5x^2 - 8cos(x)$, for which the global minimum is to be found.

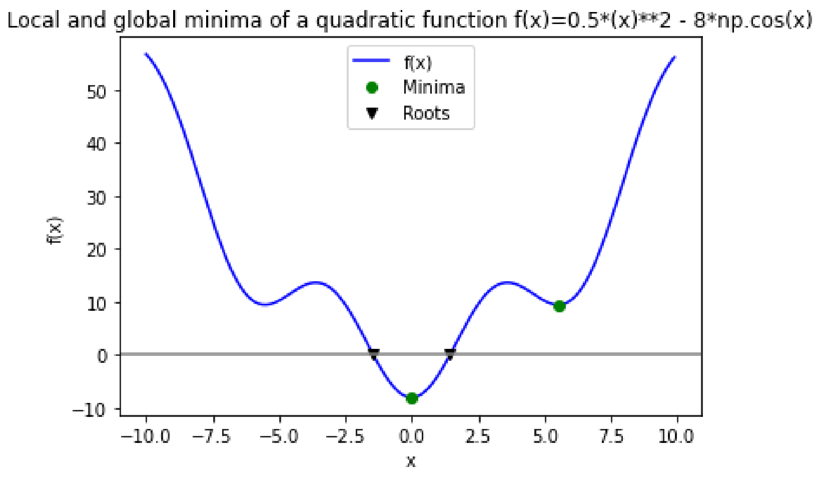


Figure 5.10: Plot showing the function f(x) with the corresponding roots and local minima

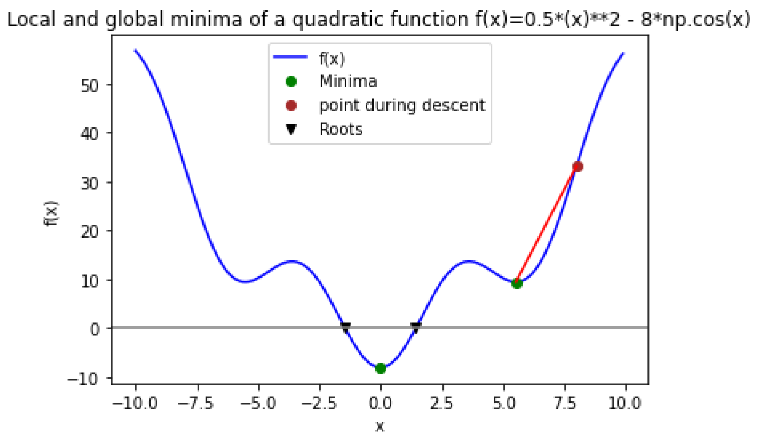


Figure 5.11: Gradient descent process during the optimization on f(x)

To proceed from the initial location represented by the red dot, a step upwards to the right is needed. As a result, the red dot is displaced from an initial point (x,f(x)) to a certain position (y,f(y)) on the plot (Figure 5.11). The line connecting x and y is given by $l(t) = \frac{f(x)-f(y)}{x-y}(t-x) + f(x)$ and could be visualized in the form of y=at+b whereas a is the slope. If the slope is too big, y is further away from x. The size of this displacement of y from x depends on a parameter  referred to as the learning rate and is used to define the displacement as follows: $y = x_0 + f'(x_0)$. If $\lambda$ is set too large, the global maximum/minimum could be missed resulting in a deviating behavior. If $\lambda$ is too small, the optimization process could take too many steps to reach the global minimum, resulting in a very slow behavior. Therefore, careful assignment of a learning rate is required to train neural networks.

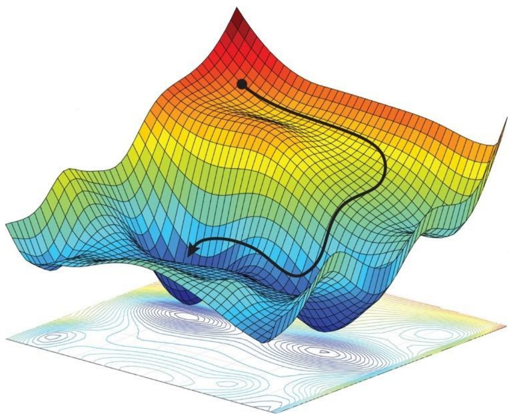


Figure 5.12: Multi-dimensional optimization process using gradient descent [44]

In contrast to this fairly simple 2D example, neural network training is often characterized by an optimization process in a much higher dimensional space. In fact, recent advances in hardware accelerator technologies paved the way for machines to work on optimization problems with parameter numbers ranging from hundreds to recent models containing millions of parameters! A better visualization of an actual deep learning optimization process is provided in Fig 5.12

Unlike the previously defined function f(x), the plot above is associated with an abstract function of higher complexity and dimensionality. Training a neural network refers to finding the global minimum for this function, which is defined by high-order range of parameters, in contrast to 2 points for Figure 5.10. The function under disposal is referred to as the loss function. In other domains, it is known under different terms such as cost function or error function (For instance, when trying to optimize the thickness of fins in heat sinks in order to maximize heat transfer). To proceed, iterative process goes in the direction of the negative gradient of this loss function until convergence occurs around its local minimum. This method is visualized in Figure 5.12 and is called the gradient descent [45].

In the sub-section below we introduce the most commonly used loss functions in Deep Learning. Let L be the loss function, y the real output and $\tilde{y}$ the model predicted output.

### 5.10.1 Commonly used loss functions for Regression

**Mean Squared Error:**

$$L(\tilde{y}, y) = \frac{1}{N} \sum_{i=0}^{N} (y - \tilde{y})^2 \tag{5.12}$$

**Mean Absolute Error:**

$$L(\tilde{y}, y) = \frac{1}{N} \sum_{i=0}^{N} |y - \tilde{y}| \tag{5.13}$$

**Hubert Loss:**

$$L_\delta(y, \tilde{y}) = \begin{cases} 0.5 * (y - \tilde{y})^2, & |y - \tilde{y}| \leq \delta \\ \delta * (|y - \tilde{y}| - 0.5 * \delta), & otherwise \end{cases} \tag{5.14}$$

### 5.10.2 Commonly used loss functions for Classification

**Binary Crossentropy:**

$$L(\tilde{y}, y) = \frac{1}{N} \sum_{i=0}^{N} [y \times log(\tilde{y}_i) + (1 - y) \times log(1 - \tilde{y}_i)] \tag{5.15}$$

**Categorical Crossentropy:**

$$L(\tilde{y}, y) = - \sum_{j=0}^{M} \sum_{i=0}^{N} (y_{ij} \times log(\tilde{y}_{ij})) \tag{5.16}$$

## 5.11 Deep Generative models

Despite the success of deep learning architectures on labeled data (supervised learning), current research trends in AI investigate whether such models are capable of ”self-learning” on unlabeled data. This research field offers several benefits, mainly the alleviating the time required to label large amounts of data. In addition, unsupervised learning in AI plays a crucial role in reducing the bias induced by human labeling processes.The most widely used generative models include Generative Adversarial Networks [46] and Variational Auto-Encoders [47]. Their ability to model complex distributions in addition to dealing with missing data and interpolation makes them an interesting candidate for numerous applications in the medical field, video tracking, and molecular chemistry [36].

### 5.11.1 Variational Auto-Encoders

Variational Auto-Encoders (VAEs) introduced by Kingma and Welling [47] are deep generative models that encode the input data into a latent space as probability distributions, in contrast to regular encoders that encode single points. Given any multidimensional input x, a VAE aims to embed the probability density of x into a latent space z through $\frac{p(x|z)\times p(z)}{p(x)}$ where p(x) is intractable and is given by $P(x) = \int P(x|z)P(z)dz$: this would be the encoding process of the data into latent representations. Yet, the integral above requires extensive computation time for a complete integral evaluation over all latent variables. To overcome this issue, we approximate the probabilistic encoder as a parametrized posterior distribution. For instance, the true posterior P(z—x) is modeled as standard Gaussian distributions and is approximated using a family of distributions q given by $q_\lambda(z|x)$. Assuming q is Gaussian, it can be represented by a set of trainable parameters $\lambda$ consisting of the approximate Gaussian distribution mean and variance, hence $\lambda_{xi} = (\mu_{xi}, \sigma^2_{xi})$ for any given data point $x_i$. Finally, the decoder through which the final reconstruction of the data is obtained is modeled as a likelihood function $p_\theta(x|z)$ parametrized by $\theta$.

To proceed, the quality of the data reconstruction by means of the VAE architecture is to be evaluated. Bearing in mind that our transformed data in latent space is a probability distribution on the one hand, and the fact that we approximate its parameters and as those of a standard Gaussian with mean 0 and variance 1, a metric that compares differences between these two distributions is needed. Numerous studies consider Wasserstein distance to be a suitable metric for such tasks [ref]. Briefly speaking, this metric is inspired from work on the optimal transport theory. It measures the cost associated with transporting components of one probability distribution to reconstruct the shape of another distribution under study in a given M-dimensional space. More details on this metric will be given in a later section. In our case, the KL (Kullback-Leibler) divergence is the most conventional metric used among recent studies on VAEs to compare the distribution of the transformed data in hidden space to a reference distribution of any choice. From another perspective, the KL divergence provides an insight on the robustness and reliability of a function q in approximating the real latent space p. Mathematically, this equation is given by:

$$KL[q_\lambda(z|x)||p(z|x)] = E_q[logq_\lambda(z|x) - E_q[logp(x,z)] + logp(x)] \tag{5.17}$$

Bearing in mind that the difference between distributions should be as small as possible, the goal is then to minimize the KL divergence by choosing the optimal parameter $\lambda$ for the function $q_\lambda(z|x)$.
Let $ELBO(\lambda) = E_q[logp(x,z)] - E_q[logq_\lambda(z|x)]$. Referring back to Eq 6.1, we get: $logp(x) = ELBO(\lambda) + KL(q_\lambda(z|x)||p(z|x))$. By Jensens' inequality, the KL Divergence is always greater than or equal to 0. Therefore, as log p(x) is a constant term, minimizing KL equates maximizing the ELBO term. The final term of the ELBO equation written with respect to our network functions is given by Eq. below whose derivation can be found in Appendix.

$$ELBO_i(\theta,\phi) = Eq_\theta(z|x_i)[logp_\phi(x_i|z)] - KL(q_\theta(z|x_i)||p(z)) \tag{5.18}$$

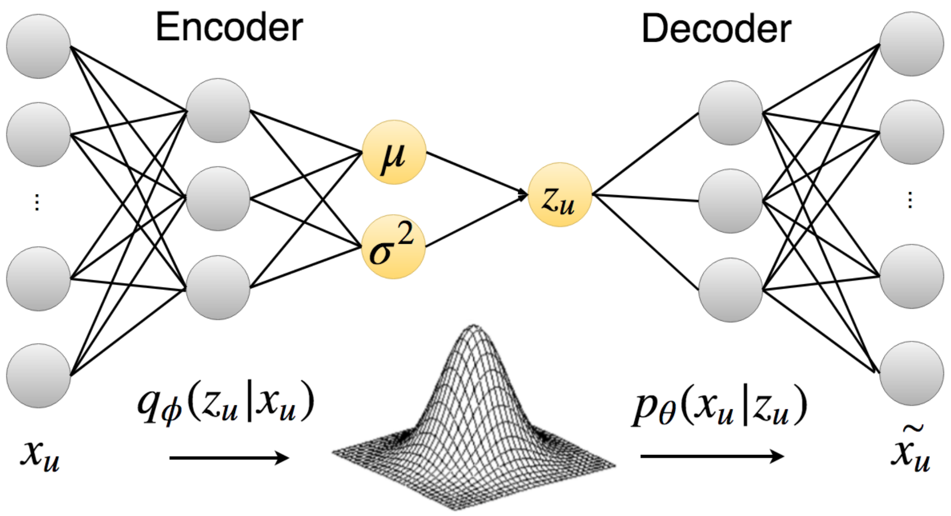


Figure 6.4: Variational Autoencoder architecture for collaborative filtering [48]

In a nutshell, a VAE maximizes the ELBO function with respect to parameters $\phi$ and $\theta$ that are optimized throughout the training process.

# Chapter 6

# Deep Learning in High Energy Physics

The success of deep learning in particle physics has been notable in recent years. Its ability to learn on complex topologies with no feature engineering while dealing with low-level information paved the way for improvements in several tasks in this field. Computer vision techniques used for object detection and image classification outperformed other techniques [10]. These techniques have been adopted in high energy physics in tasks such as jet tagging and particle identification. For instance, CNNs provided a new approach for detector data analysis through image processing where they are most successful. For this purpose, new approaches were taken with regards to LHC detector data.

## 6.1 Geometric considerations for detector-image creation

The structure of detector layers is characterized by densely packed towers forming a mesh-grid shape within calorimeters (see section 2). As a result, several studies approximated calorimeter towers as pixel grids being part of a 3D image, the latter being the entire calorimeter system. Geometric considerations are required to account for the difference in tower granularity and segmentation between calorimeter layers. In [10] , Andrews et. al suggest two approaches to tackle the difference in segmentation and overlapping regions and proceed with jet-image creation for both ECAL and HCAL. The first approach suggests an HCAL-centric geometry, whereby the HCAL segmentation is preserved while hits from ECAL endcap are projected into a grid having the same segmentation as the ECAL barrel's. As a result, a 280 x 360 image is obtained, covering the same pseudo-rapidity range as the HCAL and allowing the combination of these grid hits into one composite image. Alternatively, the second approach is ECAL-centric and consists of unrolling the ECAL barrel and ECAL endcap layers

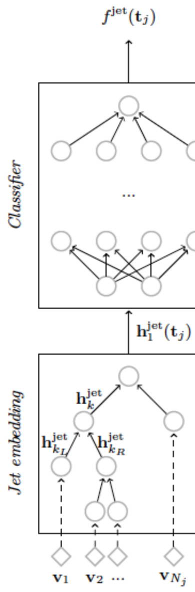

$f^{\text{jet}}(\mathbf{t}_j)$
Classifier
...
$\mathbf{h}_1^{\text{jet}}(\mathbf{t}_j)$
Jet embedding
$\mathbf{h}_k^{\text{jet}}$
$\mathbf{h}_{k_L}^{\text{jet}}$
$\mathbf{h}_{k_R}^{\text{jet}}$
$\mathbf{v}_1$ $\mathbf{v}_2$ ... $\mathbf{v}_{N_j}$

Figure 5.1: Recurrent Neural Network architecture taking jet embedding as input for jet classification [49]

into two separate images with dimensions of 170 x 360 and 100 x 100, respectively. These two approaches are adapted for the creation of ECAL images. An HCAL-centric creation of HCAL images is simpler due to the same segmentation between sections of the HCAL Endcap (HE) and HCAL Barrel (HB). Therefore, particle hits have the same location (in $\eta - \phi$ or xyz coordinates) at different depths are summed and a 56 x 72 image is created, covering both HB and HE ranges. Otherwise, an ECAL-centric approach for HCAL images projects HCAL data into an EE-like segmentation, resulting in a 100 x 100 image.

## 6.2 CNN on jet-images

In [49], Louppe et. al tried a set of approaches for jet classification. One approach is the projection of the jets' 4-momenta into images and testing their performance on MaxOut [50][51] and kt [52] architectures. This led to ROC AUC results of 0.8418 and 0.8277, respectively. The best architecture is a Recursive Neural Network that takes jet embeddings of clusters derived from the kt algorithm in [13] as input (Fig.8).This model results in a ROC AUC of 0.9222 with 100 times less parameters than the MaxOut architecture, and with 100 0000 samples compared to 6M in [12].

In [53], Andrews et. al used convolutional neural networks with a ResNet-15 architecture for classifying a jet-image as corresponding to a quark or gluon. Fol-

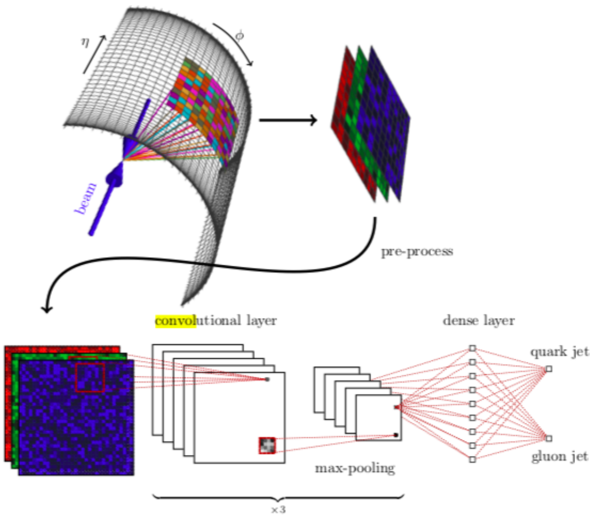


Figure 5.2: Convolutional Neural Networks on three-channeled images for jet tagging [54]

lowing image-preprocessing steps that follow the same guidelines as in [3], they obtained a ROC AUC of 80.17 outperforming the previously used recursive network in [11]. Komiske et. al [54] investigated the performance of CNNs on the task of quark/gluon jet tagging. They relied on similar geometric considerations for image processing and added 'color' the detector images by associating pixel intensities in 3D images to the transverse momentum values. For instance, red, green and blue channels composing a regular image correspond to momenta of charged particles, neutral particles and pixel-level charged-particle counts, respectively (Fig.9).As a result, more information regarding hit energy could be taken into account. Experimental results of this study show the ability of CNNs to perform equally well and sometimes better than previous jet-tagging approaches such as deep learning without color, Fisher's Linear Discriminant, shallow dense networks and boosted decision trees.

Other CNN applications in particle physics include event classification. In [55], Aurisano et.al developed a Convolutional Visual Network (CVN) were used for neutrino event detection on data collected from detectors of the NOvA experiment at Fermilab. The data consists of two types of images representing the detector's top view and side view of grid cells containing energy hits. The latter are clustered and separated from background signal caused by cosmic rays. Finally, they are fed to the CVN, paving the way for 3D event reconstruction and the neutrino event classification reaching an accuracy of 69%, further reinforcing the credibility of detector-based pixel images in high energy physics. On the other hand, deep learning techniques are used for noise reduction. Due to the high luminosity in the beam, it is unavoidable for additional parasitic interactions

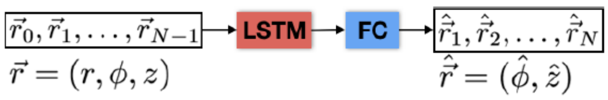

$\vec{r}_0, \vec{r}_1, \ldots, \vec{r}_{N-1}$
LSTM
FC
$\hat{\vec{r}}_1, \hat{\vec{r}}_2, \ldots, \hat{\vec{r}}_N$
$\vec{r} = (r, \phi, z)$
$\hat{\vec{r}} = (\hat{\phi}, \hat{z})$

Figure 6.3: RNN hit predictor taking 3 hit coordinates as input and outputs a 2D next-hit prediction [57]

to occur between the collision products. This rises the need for the algorithms used in analysis to account for such interactions also known as pileup which act as a noisy signal that should be separated. Similar to [56], Komiske et.al use an image-based approach and associate pixel intensities with energy distributions to build the Pileup Mitigation with Machine Learning (PUMML) model. The latter is fed with three-dimensional images with each channel corresponding to the charged LV particles, charged pileup particles and neutral particles, respectively. In contrast to previous studies shedding light on classification performance, this model was the first machine learning approach to perform regression on jet observables and returns cleaned energy distributions resulting exclusively from the leading vertex. Evaluation was done on simulated collision data generated on Pythia with pileup addition by overlaying soft QCD events. Correlation coefficients with jet observables have shown it to be a robust model performing as well as previous methods [18], giving space for further improvement on pileup mitigation using deep learning.

Image-based approaches have seen notable success for jet-substructure classification, where data contains several explorable detector features such as hit locations, hit energies and momenta. However, such approaches are hampered by the irregular grid structure of detectors on the one hand, imposing limitations on the image resolution in the data, and by the difficulty of transferring these approaches to other tasks such as particle tracking. That being said, other deep learning models in particle physics have been adopted. In [57], Farrell et.al model the dynamics of a particle trajectory using two Long Short Term Memory based models (LSTM). The first model uses a RNN that performs regression on the next hit location prediction given the current and previous coordinates (Fig.10). By minimizing a mean-squared error loss function, this model achieves reasonable results falling within 1 mm error in the z-direction.

The second model performs the same next-hit location prediction in a multimodal manner. It uses maximum likelihood estimation using a Gaussian probability distribution. The resulting prediction matches well with the actual track with a performance that is equally as good as the RNN model. Furthermore, in [58], Liechti uses RNN for track reconstruction of low-momentum particles, outperforming XGBoost with an accuracy of 87% compared to 80% with the former approach. The success of RNN in hit prediction tasks is due to its ability to account for the sequential aspect presented by particle trajectories after decays.

## 6.3 Deep Learning-based simulations

As discussed in previous chapters, Deep Learning provided notable contributions to the field of high energy physics. In fact, feeding deep neural networks with raw detector data to be processed at low-level has outperformed traditional algorithms with reliance on feature engineering and physics-based pre-processing [59]. This resulted in a surge in DL-based applications in HEP for a multitude of tasks such as jet classification [60] and trajectory prediction [58]. Nevertheless, the success of deep networks on LHC data is not limited to supervised tasks. As a matter of fact, recent work in Artificial Intelligence sheds light on the potential of deep networks architectures as unsupervised models. This could be explained by the fact that labeling data is a timely expensive task, especially when dealing with very large datasets. Therefore, the ability to train on unlabeled data provides multiple advantages including the time saved for labeling data on the one hand and alleviating human bias/errors throughout the labeling process on the other hand. Moreover, the ability of these models to train on unlabeled raw data makes them proper candidates for reconstruction tasks such as simulations. That being said, these architectures referred to as deep generative models have been widely used recently for image generation and segmentation. Recent advances in the field of deep learning and computer vision have produced notable applications in particle physics as well [61]. The potential of generative models for simulating collision events is a very active area of research. In [4], a combination of a Variational Autoencoder (VAE) and a Generative Adversarial Network (GAN) is used to simulate electromagnetic showers in calorimeters. Other studies focus on GANs for QCD Dijet events [3] and hadronic jets [62]. In our study, we make use of VAEs [47] and geometric deep learning to learn a compressed representation of the data to be used for reconstruction of high-energy physics events. As detector data in non-Euclidean by nature, we use geometric deep learning techniques including spatial graph convolutional layers [63] to learn the properties of the graph-like jets and spectral clustering layers to compress these graphs into smaller, more representative nodes.

## 6.4 Limitations

As seen in the previous section, image-based representations of calorimeter images paved the way for novel deep learning algorithms to be trained on such grid-like data format. Convolutional filters would scrutinize the detector pixels in the search for patterns specific to a unique decay event associated with a jet. Such techniques outperformed conventional algorithms and machine learning methods such as Boosted Decision Trees in numerous tasks in High Energy Physics. Despite their massive success, using convolutional networks on detector images comes with major downsides. On the one hand, most of the detector pixels undergoing convolution operations through filters are zero pixels. In fact, detector layers in CMS are very dense and it is expected that only a fraction of towers will receive the particle hits. That being said, our input data is very sparse yet very large in dimensionality whereas most of the entries are empty, resulting in a larger data size affecting the computational performance due to memory requirements. In addition, the loss function will compare the model's output to an output array with mostly zeros, hence the model can learn to output too many zeros to reduce the loss, which affects the model accuracy. In addition, in many cases, particle hits from two ore more different collision events could fuse into one common calorimeter cell. As discussed in [5], it is unclear how the CNN filters could discern the fusion of non-additive quantities such as the particle types into one single cell. This issue worsens when dealing with unsupervised data for representation learning. Finally, one last downside of image-based approaches for CNN is the limitation associated with the complex detector granularity. In fact, the detector layers' resolution is hardly homogeneous. This is especially the case for regions that overlap between the barrel and endcap section (See Chapter 3) where a fixed-size $(\eta, \phi)$ resolution of $(0.025 \times 0.025)$ highly undermines the complexity of the geometry at those locations. As discussed in [53], ECAL and HCAL detectors are characterized by different segmentations, especially in the endcap regions, hence posing a challenge for the construction of high-fidelity detector images.

In a nutshell, CNNs provide a promising performance on particle physics tasks, especially when mapping event data into images where their performance is maximal. Yet, mapping detector data into images of given resolutions is problematic due to the sparse nature of the events within calorimeter cells and the latter's heterogeneous and complex detector geometry. As a result, there is a need to map these convolution operations into hit locations under consideration, with disregard to the numerous empty cells surrounding them.

# Chapter 7

# An alternative approach, Graph Networks

Deep learning techniques have notably shaped the analysis of several types of data including images, videos and speeches. In addition, the previous sections have shown how these techniques are multi-disciplinary, as they were transferable to particle physics where they set the ground for several explorations. The latter inspired from computer vision techniques and used different network architectures to unravel new physics. In spite of alleviating the cost associated with feature engineering adapted in classical machine learning algorithms, neural network architectures such as CNNs still have limitations to consider. These include information loss due to geometrical considerations on detectors that perform data projection on detector layers' grids with reference granularity. In addition, CNNs require a set of steps for image pre-processing that could be lengthy and computationally costly as they have a high number of trainable parameters. Therefore, data is frequently transformed to be treated as a regular structure in the Euclidean domain. Nevertheless, numerous types of data are irregular in shape, which raises the need for new approaches to analyze data having a non-Euclidean geometric structure. That being said, graphs are an example of a data type whose structure is suitable to represent various complex problems involving interactions between entities. As a result, recent studies have shed light on the potential of Geometric Deep Learning (GDL). Based on graph theory, GDL aims to transfer deep learning techniques, mainly convolution operations and pooling, to irregular-structured data such as graphs and manifolds [64] [65].

This method has been successful in a multitude of applications. In ecommerce, graph architectures have shaped the way user-product interactions occur with the recent advances in recommender systems, allowing the use of graphical representations where edges represent similarities between items defined as nodes [66]. In sciences, graph networks provided a new platform for molecular and drug research, where graphs represent molecular compounds whose interacting molecules

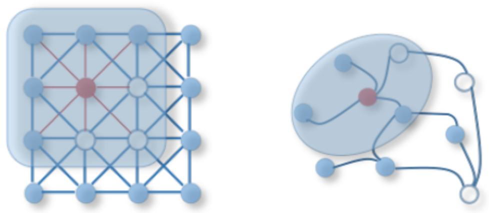


Figure 7.1: 2D Convolutions vs Graph Convolution [64]

through one or more bonds are defined as nodes and edges, respectively [26]. Finally, a commonly example of graph networks in the DL community uses the CORA citation networks dataset, where nodes are scientific papers whose neighbors form the list of citated work in this paper [26].

In the section on Deep Learning, we mentioned how CNNs make use of the image data's locality and shift invariance, allowing convolution operations to extract features in a Euclidean domain through sliding kernel filters. This approach is not feasible in the non-Euclidean domain because the local connectivity differs between entities; i.e different entities could be connected to a different number of neighbors. At this point onwards, we define these entities as nodes (or vertices) whose connectivity is defined by a set of edges whose architecture is pre-defined with an adjacency matrix.

## 7.1 Definition

A graph is denoted by $G = (V, E, A)$ where V is the set of vertices composing the graph, E the set of edges connecting these vertices, and A being an N x N adjacency matrix, where N is the total number of nodes in the graph. Associated with each vertex V is a set of features describing it. These are given in a feature vector $X \in R^{N \times D}$ with D being the number of features per node.

Let $v_i \in V$ denote a vertex and $e_{ij} = (v_i, v_j) \in E$ an edge connecting this vertex to a neighbor $v_j$. $A_{ij}$ then denotes the value $w_{ij}$ at the $j^{th}$ column of the $i^{th}$ row of the adjacency matrix. In the case of a weighted adjacency matrix, $w_{ij}$ represents the weight of an edge connecting two vertices, indicating the effect presented by $v_j$ on $v_i$'s features as compared to other neighbors. An un-weighted adjacency matrix is characterized by a $w_{ij}$ value equal to 1 if a connection exists between two vertices, and a value of 0 otherwise. The total sum of neighbors for a node $v_i$ represents the degree of this node and is given by $\sum_{j=0}^{n} w_{ij}$ where n is the number of neighbors. At this stage, we distinguish two types of graphs: A directed graph is one where the edges unidirectionally point from one vertex to the neighboring one. As a result, it is likely that $w_{ij} \neq w_{ji}$ in such a case. In contrast, connectivity and edge features go both ways in an un-directed graph hence $w_{ij} = w_{ji}$ [21].

## 7.2 Graph Networks

Kipf et.al [67] presented one of the first attempts to perform convolution on graphs. The concept behind Spectral Graph Convolution is inspired from Graph Signal Processing whereas a feature vector X associated with the node attributes of a graph G is treated as a signal in the frequency domain. In order to perform convolution on graphs in the spectral domain, the first step consists of calculating the normalized graph Laplacian given by:

$$L = I_n - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U \tag{7.1}$$

In Eq, $I_n$ is an identity matrix, $D \in R^{nxn}$ is the diagonal degree matrix given by $D_{ij} = \sum_{j=0}^{n} w_{ij}$, $\Lambda \in R^{nxn}$ is the diagonal matrix of eigenvalues of eigenvalues of L where $\Lambda_i i =$ and $U \in R^{nxn}$ is the graph Fourier basis, a matrix containing the eigenvectors of L ordered by eigenvalues.

At this point, eigendecomposition of the graph Laplacian matrix has been done, hence factorizing it into as many eigenvectors as the total number of nodes in the graph. The convolution operation can now be performed using a spectral kernel that would be multiplied by the node signals. In [24], Hammond et. al suggest that the kernel on $\Lambda$ can be formulated as a truncated expansion in terms of Chebyshev polynomials, as shown in the following formula:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda}) \tag{7.2}$$

where $\tilde{\Lambda} = \frac{2}{\lambda_{max}} \Lambda - I_n$, $\lambda_{max}$ being the largest eigenvalue in $\Lambda$.

The kernel $g_\theta$ is applied to the diagonal matrix of eigenvalues. $\Theta$ is a vector containing Chebyshev coefficients to reduce the number of parameters, T contains Chebyshev polynomials and K is the order of neighborhood away from a given node to be covered.

## 7.3 The propagation rule

Having performed the convolutions operation on neighbor node attributes, this signal needs to be aggregated and propagated k times in order to update the features of the node under study for the next iteration. A hidden GCN is given by $H^i = f(H^{i-1}, A)$ where $H^i = N \times X^i$ represents the node features at iteration i and f is a propagation function, most commonly the ReLU function. The simplest form of the propagation rule accounts for the node features by summing them prior to aggregation. This method known as the sum rule is given by: $h_{vi} = A_i X = \sum_{j=1}^{N} A_{ij} X_j$ where $h_{vi}$ represents features of node v at the $i^{th}$ iteration. Nevertheless, one problem arising from this method is its inability to account for self-features i.e the features of the node under consideration. The mean rule addresses this issue using a self-loop by adding the identity matrix to the adjacency matrix $\hat{A} = A + I$. The mean rule is given by:

$$h_{vi} = D^{-1} A_i X = \sum_{k=1}^{N} D_{i,k}^{-1} \sum_{j=1}^{N} D_{i,k}^{-1} A_{i,j} X_j = \sum_{j=1}^{N} D_{i,i}^{-1} A_{i,j} X_j = \sum_{j=1}^{N} \frac{1}{D_{i,i}} A_{i,j} X_j = \sum_{j=1}^{N} \frac{A_{i,j}}{D_{i,i}} X_j \tag{7.3}$$

where D is the diagonal degree matrix where the $i^{th}$ row and column represent the degree of node i. Since all other entries are 0, the summation over k is irrelevant and can thus be removed.

One last formulation accounts for both the degrees of node i on the one hand and the neighbor nodes' on the other hand. The spectral rule formulation is given by:

$$h_{vi} = D^{-0.5} A_i D^{-0.5} X = \sum_{k=1}^{N} D_{i,k}^{-0.5} \sum_{j=1}^{N} A_{i,j} \sum_{l=1}^{N} D_{j,l}^{-0.5} X_j = \sum_{l=1}^{N} D_{i,i}^{-0.5} A_{i,j} D_{j,l}^{-0.5} X_j \tag{7.4}$$

where the third summation over l aggregates features from the nodes that are directly connected to the neighbor j of node I under study. This rule helps to avoid overfitting in case some nodes have notably higher or lower degrees than others.

## 7.4 Graph pooling

As seen in Chapter 4, pooling operations on Euclidean data are successful in reducing the number of trainable parameters and learning compressed representations of images. Given the dense nature of some graph such as social networks characterized by thousands and sometimes millions of nodes, a similar operation in the non-Euclidean domain is needed. Having said that, recent studies have shed light on several approaches that use pooling in graph neural networks. Early studies on graph pooling make use of a graph's topology to learn clusters of it through conventional graph coarsening algorithms that perform eigendecomposition in the Laplacian domain. This clustering method is referred to as Graclus and has been used in several GNN applications [68] [69]. Later methods make more use of node features through global pooling such as the SortPool layer introduced by Zhang et.al in [70]. SortPool embeds an unordered set of nodes vertices after several GCN operations and arranges them in a sorted fashion of fixed size prior to feeding them to a 1-D CNN and a fully connected network for classification. Despite setting a good baseline for graph and node classification using pooling, global pooling fails to account for the hierarchical aspect between the nodes and the fact that some clusters of nodes are more representation of the overall graph than others. These drawbacks were tackled by Yhing et.al in [71] where they introduce DiffPool, a hierarchical graph pooling that, in contrast to graclus, operates on an assignment matrix S by learning on it. More specifically, DiffPool assigns to each node within a graph of $p$ clusters at layer $l$ a probability of belonging to one of $q$ clusters in layer $l + 1$ where $q \leq p$. Other studies on graph pooling disregard clustering through assignment matrices and proceed by assigning relevance scores by means of attention layers whereas we selected the top-k nodes or edges as is the case with SAGEPool and EdgePool, respectively [72] [73].

## 7.5 Graph Variational Autoencoders

The potential of Variational Auoencoders on non-Euclidean data in deep generative model applications remains under-explored. Variational Graph Auto-Encoders (VGAE) were first introduced by Kipf et.al in [74] for unsupervised learning-based tasks such as link prediction on the following citation network datasets: Cora [75], Citeseer and Pubmed. This was a crucial architecture that facilitated the analysis of graph structures with large numbers of nodes, mainly social networks and citation networks. The latter could be trained in a more computationally efficient manner while still giving state-of-the-art accuracy. For instance, VGAE and a regular GAE outperformed conventional graph kernels such as DeepWalk [76] and Spectral Clustering [77] in the link prediction task with accuracies of 91.4 and 91 % for the VGAE and GAE, respectively.

Given their notable success in network mining, a multitude of researchers have shed light on the potential of graph-based architectures in their area of specialty. Today, graph generative models are the core of molecular chemistry and drug research. This could be explained by the fact that molecular compounds are defined by an explicit non-Euclidean/graph-like topology. Hence, researchers were motivated to map a given molecular structure into a graph topology to be used in a deep graph generative model for several tasks such as molecule classification [ref], property prediction [36] and novel constrained molecular shape generations [37]. In [37], Simonovsky and Komodakis develop a GraphVAE model for molecular graph generation using 2 datasets: QM9 [78] and ZINC [79]. The nodes (atoms) are embedded into latent space vector representation approximating an isotropic Gaussian prior distribution $N(0, I)$. Then, their probabilistic graph decoder outputs a fully connected graph with k nodes where k is a pre-defined maximum number of nodes within a graph. Their conditional VGAE outputs a limited number of valid chemical compounds, yet remains a major step towards incorporating deep generative models in molecular chemistry.

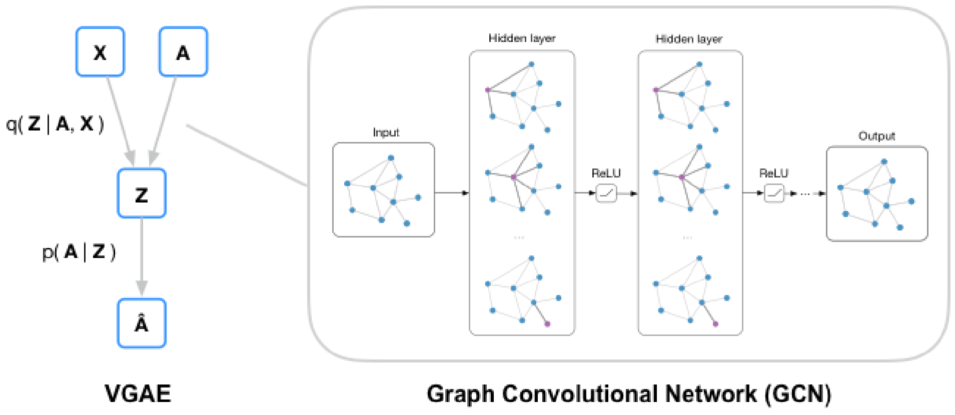


Figure 7.3: Graph Autoencoder architecture [80]

## 7.6 Graph Networks on Point Clouds

Rather than mapping any sparse structure defined by coordinates in a M-dimensional space into a multi-channel image to be fed to a fully-connected or convolutional network, such structures could now be treated as point clouds in M-dimensional space. Recent studies have shed light on the potential of deep learning in incorporating such sparse representations for numerous applications, mainly classification, shape retrieval and part segmentation. To proceed, the notion of Permutation Invariance in point clouds must be highlighted. For instance, training neural network models on images requires the former to be invariant or "resilient" to any rotations and augmentations of these images. Similarly, point cloud rotations and node permutations should not affect the model output. In [81], Qi et. al developed PointNet, a deep learning model that accounts for the permutation invariance of points within their 3D representation to perform classification and segmentation tasks. More specifically, given a sample of N points in 3D space, PoinNet is invariant to $N!$ permutations of this sample. Eventually, this model was able to obtain a mean Intersection-Over-Union (mIOU) of 83.7 on the ShapeNet part dataset [82], outperforming all baselines. Later on, Qi et.al introduced an upgraded version, PointNet-++, whereas local structures can be discerned in the metric space by performing PointNet recursively on overlapping local regions of point clouds to extract higher level features [83].Eventually, PointNet-++ reaches a mIOU of 85.1

Having introduced the early deep learning techniques on point clouds, we now shed light on the incorporation of graph neural networks in such applications, and the added value they present. In [84], Wang et.al introduced the Dynamic Graph CNN, a novel architecture that learns on raw 3D data by means of Edge Convolution (EdgeConv) layers. These layers operate in a dynamic fashion on point clouds mapped into a graph topology defined by node features,edge features and an adjacency matrix. The latter is defined by means of k-nearest neighbours for each node in 3D space at input level, and in latent space within hidden layers. This architecture was trained on ModelNet40 [85] for classification, reaching a mean class accuracy of 90.7% and an overall accuracy of 93.5%. In addition, DGCNN was trained for part segmentation on the Shapenet part dataset reaching a mIOU of 85.2, outperforming all baselines except PointCNN [86].Another scenario where graph convolution is performed on 3D point clouds is shape reconstruction through generative modeling (more details in Section 7.4). For instance, Bouritsas et. al [87] make use of GNNs in learning latent representations of 3D meshed topologies of human faces. The latter are encoded into latent space through a sequence of graph convolution and pooling operators. Their Neural3DMM model then smoothly reconstructs the meshes from latent space.

The aforementioned techniques for learning on point clouds were crucial for the advancement of deep learning research on non-Euclidean structures, which are the most dominant forms of data around us. For instance, their ability to incorporate raw data into graph architectures and applying spatial convolution solely on the nodes and edges under consideration makes them a suitable replacement for imaged-based CNN techniques in terms of memory requirements and computational complexity.

## 7.7 Graph Networks in High Energy Physics

Having discussed the applications of graph-based architectures on point clouds, we now emphasize on the promising aspect of such methods in particle physics. Several attempts have been recently made to approach particle collision events spread within detectors as point clouds mapped into graph topologies. This technique is memory-efficient as it operates directly on the raw data associated with particle hits' locations in the calorimeter and their respective momenta. Consequently, a multitude of GNN applications surged in HEP. In [5], Qu and Gouskos introduce ParticleNet, a deep learning architecture with similar properties as DGCNN in [84] for jet tagging purposes. In this study, they perform classification using the Top jets [88] and Quark/Gluon datasets [89], respectively. To proceed, their model is fed with 4 main features of jets including the $\eta - \phi$ coordinates of the hits, the PID and their respective hit energies. Next, k-nearest neighbor approach is used to connect each node (particle) to k neighbours around it according to its $\eta - \phi$ coordinates: this defines the graph topology of the jet. Finally, the input is propagated through EdgeConv layers along which the graph topology changes depending on node coordinates in latent space. Eventually, ParticleNet outperforms baselines including ResNetXt-50, P-CNN and PFN in both top tagging and quark/gluon classification with AUC scores of 0.9858 and 0.9116, respectively. Yet, one downside of ParticleNet is its relatively slow inference time on GPU (0.92 ms compared to 0.018 ms for PFN). This is due to the slower nature of EdgeConv layers.

Other graph-based applications in HEP include Pileup Mitigation [90] and particle reconstruction [91] [92]. In[90], Martinez et.al make an extented version of the PUPPI algorithm (PileUp Per Particle Identification) [93] using deep learning and graph networks. Their model called PUPPIML takes as input several particle features, mainly their coordinates, charge and probability of being a pileup (for a detailed description refer to [90]). The input mapped into a graph which is fed to 3 consecutive Gated Graph Neural Network (GGNN) layers. Eventually, a fully connected network processes the embedding of individual nodes and outputs a binary pileup classification score. The GGNN architecture outperforms baselines such as PUPPI, FCN, GRU and the SoftKiller algorithm in pileup classification performance (ROC of 96.1%). This technique is efficient on the node level and is similar by analogy to graph network architectures used for node classification tasks in social networks and citations networks whereas the algorithms are used for recommendation of products or classification of articles, respectively.

On the other hand, Ju et.al [91] make use of GNN modules based on Interaction Networks from [94] to perform particle track reconstruction in calorimeters of complex geometries. They pre-process data belonging to the barrel region of the detector only (See Chapter 2) by assigning cylindrical coordinates $(r, \phi, z)$ as node features and the differences between them $(\Delta\eta, \Delta\phi)$ as edge features. To proceed, they encode the input graph's node and edge features into latent space through two fully connected networks. Next, they perform N iterations of graph layers to learn the interaction between particles in latent space (N=8 was found to perform best). Eventually, a decoder with two fully connected networks reconstructs the particle track. Their model resulted in an overall relative efficiency of 95% for track finding.

## 7.8 Summary

From the aforementioned studies, we can conclude that Graph Neural Networks show a notable potential on point clouds on the one hand, then in the field of HEP on the other hand. Their ability to study interactions between interconnected nodes in 3D space and any latent space makes them a strong candidate for the study of collision events obtained at the CMS detector. That being said, this thesis work will shed light on the ability of graph-based architectures to perform a set of tasks such as fast simulation of boosted jets from the parton level on the one hand and the estimation of muon momenta at the level of Cathode Strip Chambers on the other. Further details are discussed in the next chapters.

# Chapter 8

# Falcon

Particle collisions taking place at the LHC are very complex in nature. Hence, several attempts have been made to reconstruct collision events through the development of advanced probabilistic models and Monte Carlo techniques. A probabilistic model of the event reconstruction problem could be described as follows:

$$\begin{aligned} p(\text{r-particles}|\theta) = \int & R(\text{r-particles}|\text{particles}) H(\text{particles}|\text{partons}) \\ & \times P(\text{partons}|\theta)\ d\text{particles}\ d\text{partons} \end{aligned} \tag{8.1}$$

where p represents the probability density of observing a set of reconstruction particles given a point in the parameter space and Rparticles is the detector response [95]. The latter, as previously mentioned, could be approximated in different ways depending on the type of simulation being performed; either a Full simulation using Geant [1][96], or a parametric simulation using the DELPHES framework [2]. The former is an interactive software toolkit used in HEP to simulate the interaction of particles with matter while accounting for several boundary conditions such as detector geometry, magnetic field and other physical aspects. Despite their accurate representation of collider events, Geant-based simulations are highly complex and require extensive computing resources, bearing in mind that higher amounts of data need to be collected from the LHC in the future. As a result, Fast Simulation techniques have been introduced as a computationally efficient and faster alternative at the expense of complexity. DELPHES framework has been introduced as a toolkit for fast detector response simulation whereas the environment consists of the trackers, calorimeters and muons systems wrapped around the beam axis. Users are able to specify the segmentation of the calorimeters on the one hand, and the strength of the applied magnetic field on the other hand. Eventually, this software allows us to perform numerous tasks such as jet reconstruction originating from a photon/electron or the calculation of missing jet transverse momenta. In a nutshell, DELPHES is a C++ - based code that uses simplified detector geometries to simulate particle interactions with matter, hence mapping the detector response into a parametric function. Nevertheless, a major issue with this method is its non-universality. For instance, each simulation is unique to the underlying detector geometry under consideration, hence the need to hand-code the framework accordingly with any required detector changes. In this work, we shed light on new methods that utilize non-parametric methods to simulate the detector response function. The first such application was led by Ted Knuteson through the development of the TurboSim program in [97]. Our Falcon project could be considered as an updated version of TurboSim and its package consists of two main components: the builder and the simulator. The former is used to derive a formulation for the response function through the analysis of fully simulated events. On the other hand, the simulator uses the builder's formulation of events at the parton level to simulate the resulting events at the reconstruction level. In probabilistic terms, the simulator estimates the product given by:

$$R(r - particles|particles) \times H(particles|partons) \tag{8.2}$$

The end-result of the simulator will be a jet corresponding to different types of flavors such as electrons, muons and taus. A proof of principle in addition to further information on FALCON's early work can be found in [95].

The above mentioned models have shown early success in representing detector data. For instance, DELPHES has shown promising results in several tasks such as Jet reconstruction, missing transverse momenta calculation and Pileup mitigation using the Jet Area method [2]. Figure 8.1 shows the energy resolution resulting from the DELPHES simulation of electron and photon showers as compared to the original resolutions of these showers at the CMS detector. Considering the rising need for scalability of any given model to cope with the larger amounts of data to be produced in LHC at higher luminosities, the search for computationally efficient and data-friendly algorithms continues. That being said, the next section will shed light on the promising aspect of Artificial Intelligence and Deep Neural Networks in Particle Physics, and their ability to learn the properties of collider events' data to perform a variety of tasks using both supervised and unsupervised methods. Eventually, we explain how FALCON makes use of these novel techniques in Fast simulation applications and how these methods are more efficient than the traditional Monte Carlo-based models.

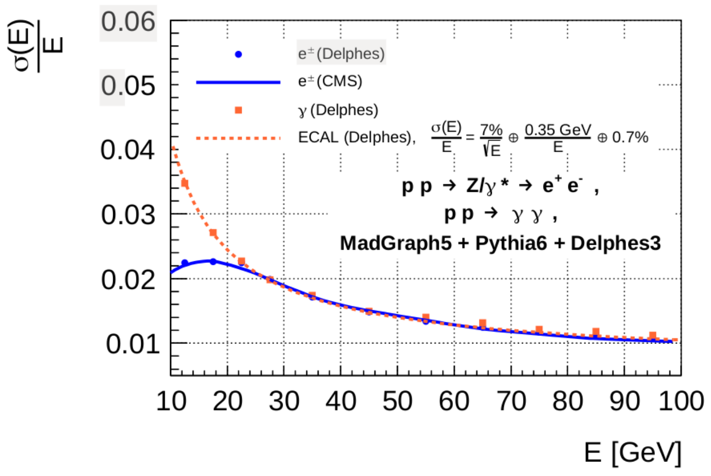


Figure 8.1: The energy resolution resulting from the DELPHES vs original resolution at the CMS detector [2]

# Chapter 9

# Methods and Results

## 9.1 Data

### 9.1.1 Definition

In this work, we make use of the CMS Open Data release [98] - publicly accessible data from the LHC experiments. We consider the boosted top quark jets produced using Pythia 6, a program for generating particle collisions events. The data was transformed into image-based form, specifying the location and values of energy deposits in the calorimeter by following the prescription in [60]. The data consists of almost 30000 samples of 3x125x125 arrays representing the mesh and the segmentation of 3 detector stages: Tracker, ECAL and HCAL subdetectors, respectively. We aim to reconstruct jets that deposit their energy in the calorimeters, initially focusing only on the ECAL subdetector hits. The non-zero hits within this 125x125 array correspond to the hit energy of the corresponding particle shower deposited at that specific grid cell.

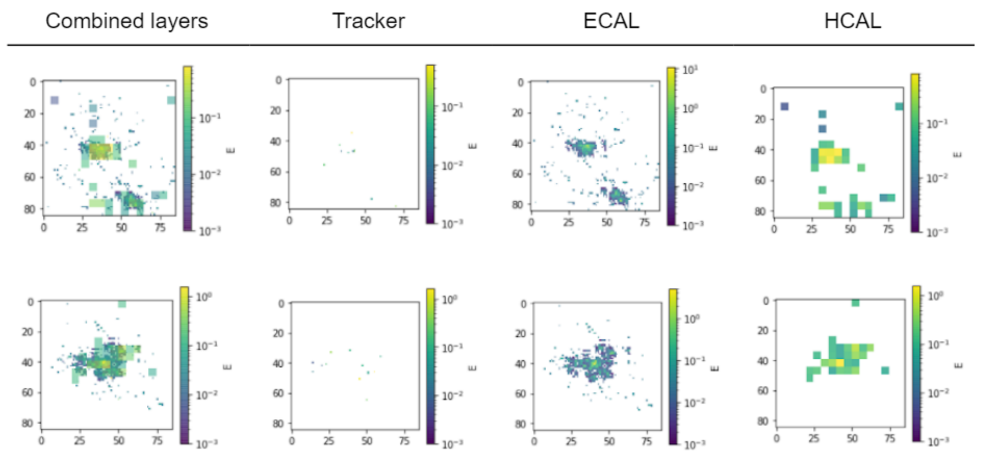


Figure 9.1: Comparison between the energy deposits in the Tracker, ECAL, and HCAL calorimeter layers.

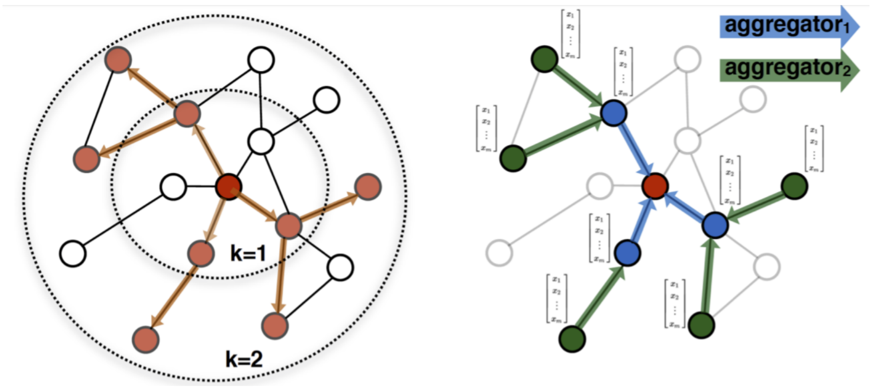


Figure 7.2: GraphSAGE message passing through neighborhood feature aggregation[99]

### 9.1.2 Pre-processing

We pre-process the data by selecting the non-zero hit locations within the array, providing their respective x and y locations as per the calorimeter segmentation. Afterwards, we concatenate the x,y locations with their corresponding hit energy at that location. At this stage, each sample has the shape Nx3 where N is the number of non-zero particle hits within the detector for one specific sample jet, with each sample containing 3 features: the x,y locations and their hit energies, respectively. In the next section we show that N is also the number of nodes within one graph representing a jet.

## 9.2 Use Case: GraphSAGE

In our work we are interested in embedding node representations of a graph into a latent space. As our point cloud data of physics jets does not have a unique pre-defined topology, it is possible that propagating messages within direct neighbouring nodes is not enough to generalize the description over the whole network. Therefore, we refer to GraphSAGE [99], an inductive learning framework that assumes nodes within the same neighbourhood to have similar embeddings and proceeds by aggregating information from higher-level neighbourhoods. The depth of the latter is described by a constant K, whereas K=2 refers to two-hop neighborhood relative to the node under study. Furthermore, in contrast to conventional spatial convolutions where message passing is performing within an entire neighbourhood, GraphSAGE samples random nodes from the given neighbourhood and aggregates and their latent space features. The latter are then concatenated with the node's initial embedding at that same iteration and sent to an activation function. Therefore, GraphSAGE is a memory efficient deep learning algorithm that makes a suitable candidate for a graph encoder-decoding process of physics jets given its ability to learn on long-range neighborhoods.

**Algorithm 1** Multi-layer GraphSAGE pseudo-code

```
0: Input: Graph G=(X,A) where X is the feature matrix and A the adjacency
   matrix in the model's $l^{th}$ layer; X $\in$R$^{n\times d}$ and A $\in$R$^{n\times n}$ with n=number of
   nodes in a graph, d=feature dimension.
   Let $u$ be a node's features and $\mathcal{V}$ be the neighbourhood of this node.
0: for $l = 1, 2, \ldots, L$ do
0:     for $v \in \mathcal{V}$ do
0:         $h^{L}_{\mathcal{N}(v)} \leftarrow \text{aggregate}_l(h^{l-1}_u, \forall u \in \mathcal{N}(\mathcal{V}))$
0:         $h^{l}_{v} \leftarrow \sigma(W^l.CONCAT(h^{l-1}_v, h^{l}_{\mathcal{N}(v)}))$
0:     end for
0:     Normalize the feature vectors prior to message passing:
0:     $h^l_v \leftarrow \frac{h^l_b}{\|h^l_v\|}$
0: end for
0: Aggregate hidden features to original node:
0: $h_{v+1} \leftarrow h^l_v, \forall v \in \mathcal{V}$ =0
```

## 9.3 Mincut Pooling

In this work we make use of Mincut Pooling layer used by [100] to compress graph representations. Given a graph $G = (N, E)$ with N being the number of nodes and E the number of edges. Mincut Pooling operates by distributing the graph nodes into clusters with similar latent features by means of a cluster assignment matrix $S \in \{0, 1\}^{N\times K}$ with $S_{i,j} = 1$ indicating the inclusion of node i in cluster j. In mathematical terms, this process is described by [101] as follows: $argmax_{Q\in R^{N\times K}} = \frac{1}{K}\sum_{k=1}^{K} Q_k^T A Q_k$ having a solution $Q^* = U_K O$ where $U_K$ is the matrix of eigenvectors ordered by the K largest eigenvalues. Graph spectral clustering learns a cluster assignment matrix S by linear transformations of the node representaions in latent space followed by softmax activation for class/cluster assignment. This method clusters nodes according to the graph topology and node features; nodes that have similar features or are closely connected should have similar representations in latent space and are likely to fall within the same cluster. To proceed, the pooled feature matrix and adjacency matrix are calculated as follows:

$$A^{Pool} = S^T \tilde{A} S \tag{9.1}$$

$$X^{Pool} = S^T X \tag{9.2}$$

## 9.4 Model architectures

In this study, we make use of geometric deep learning methods where convolution operations are translated to non-Euclidean structures, in contrast to previous studies where the entire grid-like structure of a given detector is used as an input to fully-connected or convolutional layers for classification or regression purposes [54][55]. Therefore, we consider particle hits within a detector to be interconnected nodes in a graph. In contrast to molecular chemistry, where the graph topology is constrained by the molecule shape [37], jets in particle collisions are not characterized by such pre-defined topology. We proceed by connecting each node to its k-nearest neighbours based on Euclidean distance given by $\sqrt{(x-x_i)^2+(y-y_i)^2}$ with $x_i$ and $y_i$ referring to this node's coordinates. In this thesis, we present two methodologies for jet reconstruction through decoding from latent space. The first method, GVAE-Mincut,consists of embedding the graph's node features into latent space through SAGE layers while pooling the graph nodes to get a compressed representation of this graph. The second method,GVAE-EdgePool uses a simpler architecture whereas node features are embedded into latent space without pooling. In latent space, we obtain a hidden representation with the same number of nodes and different number of features which are in vectorized form. On the one, we compare the performance of our model to that of a graph model that uses EdgePooling. Next, we compare the reconstruction and number of parameters of both graph models with a regular autoencoder that operates on images. Finally, we check for the speedup obtained by using deep generative models as compared to traditional Monte Carlo techniques.

### 9.4.1 Jet Reconstruction with Mincut pooling

To learn the properties of these jets as graphs in addition to a compressed representation to be used in an encoder-decoder architecture, we develop a Graph VAE architecture whose encoder embeds the node features into latent space dimensions through Dense GraphSAGE layers [99], then compresses them into smaller dimensions using dense mincut graph pooling operations inspired by [100] where spectral clustering of the graph nodes is performed. A pictorial representation of our model is given in Figure 9.2. The latter shows the process of sampling non-zero hits from grid arrays and mapping them into graphs, resulting in an adjacency matrix A and a feature matrix X. Both matrices are used as input to a total of 3 Dense GraphSAGE layers within the model's encoder of, each of shape $(batch\ size, n_{nodes}, n_{features})$. Amid those GraphSAGE layers are two Dense Mincut Pooling layers that downsample the graphs' nodes to smaller dimensions. Furthermore, the compressed graph representations are linearly transformed into a final latent space layer at which stage they undergo reparametrization to allow backpropagation to take place. Finally, a decoder performs decoding of the latent space compressed nodes to obtain upsampled feature matrix X and adjacency matrix A, respectively as follows:

$$X^{rec} = SX^{Pooled}; A^{rec} = SA^{Pooled}S^T \tag{9.3}$$

where S is a learned cluster assignment matrix similar to the one defined in [100].

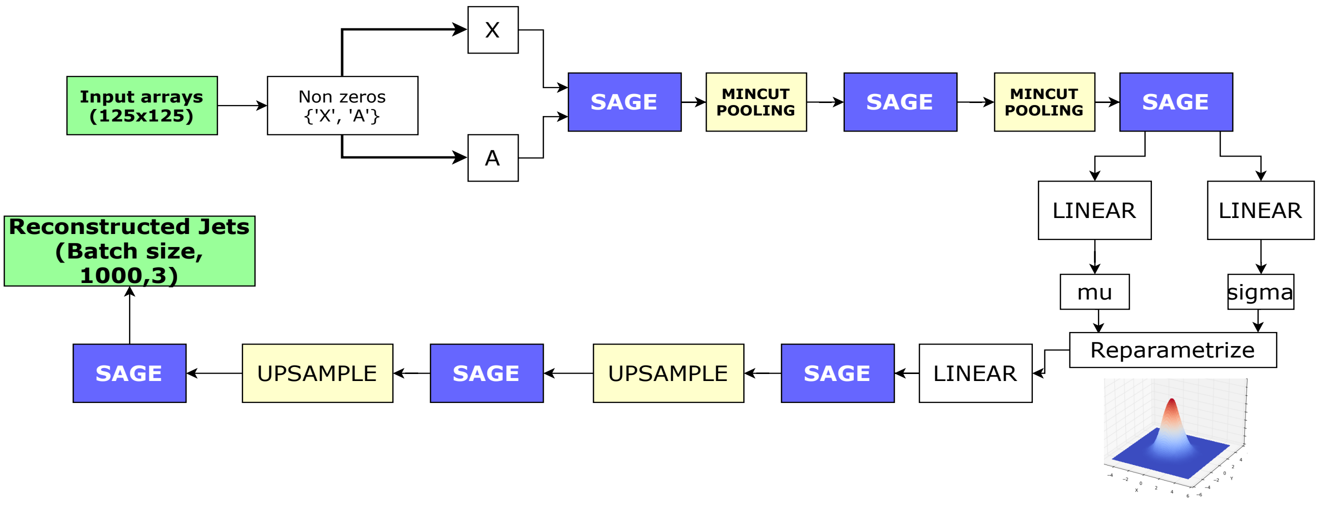


Figure 9.2: Model architecture of the Graph Variational Autoencoder showing GraphSAGE layers and pooling blocks.

To proceed with training, we choose k=4 as the number of nearest neighbours to be connected to each node. In addition, we use the Adam optimizer with a learning rate of 0.001. Finally, our loss function includes the MSE loss between node features on the one hand, and the Kullback-Leibler divergence between the latent space and the real P(z|x) distribution. Prior to training, we split our dataset into 70% training, 20% validation, 10% testing.

### 9.4.2 EdgePooling

We compare the performance of our Mincut GVAE with an EdgePooling based autoencoder. This model learns a hidden representation of jets as graphs by performing message-passing within the nodes i.e spatial GCN and by compressing their large representation by means of localized pooling transformations as described by [73]. Similar to attention mechanisms, transformed node features are first concatenated and undergo a tanh activation which computes node scores.The latter are then normalized by means of a softmax activation.Eventually, we obtain a sorted list of edge scores from which the top k% are selected where k is the percent of graph reduction to perform.

Through both GVAE methods (Mincut and EdgePooling-based), we use the tanh activation function in the first layer of the encoder, with the rest of the layers using ReLU activation afterwards. In addition, adding batch normalization seems to stabilize training further, so it was added within encoder layers.

## 9.5 Metrics

First introduced in 2019, the Earth Mover Distance (EMD) metric describes the space of two collider events [102]. It represents the minimum "work" needed to be applied by the movements of energy $f_{ij}$ from particle $i$ in one event to particle $j$ in the other so that event $\mathcal{E}$ is rearranged into event $\mathcal{E}'$. In other words, given two probability distributions, the EMD tells how costly it is to reshape all elements of on distribution (in our case a point cloud) to become similar to the other one, in analogy to the physical work unit in Joules used in classical physics. In mathematical terms, given two distributions or point clouds I and J, the EMD value can be computed as follows:

$$EMD(I,J) = \min_{f_{ij}} \sum_{i \in I} \sum_{j \in J} f_{ij} \frac{R_{ij}}{R} + |\sum_{i \in I} p_{T_i} - \sum_{j \in J} p_{T_j}| \tag{9.4}$$

where R is the radius of the jet event, $R_{ij}^2 = (y_i - y_j)^2 + (\phi_i - \phi_j)^2$ is the rapidity-azimuth distance. In addition, $f_{ij}$ represents the quantity of work required to transport one particle from jet I to its compatible location within jet J. This transport is constrained by the following condition:

$$f_{ij} \geqslant 0, \sum_{j \in J} \leq p_{T_i}, \sum_{i \in I} f_{ij} \leq p_{T_j} \tag{9.5}$$

$$\sum_{i \in I} \sum_{j J} f_{ij} = \min(\sum_{i \in I} p_{T_i}, \sum_{j \in J} p_{T_j}) \tag{9.6}$$

We therefore use the EMD metric to assess the quality of reconstructed jets produced by our fast simulation.

## 9.6 Results

We obtain our results after training on a Tesla P100 GPU provided by the Google Colaboratory framework.

### 9.6.1 Graph generative model with mincut pooling

In Figure 9.5 we show the reconstruction result for several simulated jets from our Mincut GVAE model. The plots show that our model's decoder is able to accurately reconstruct the jets from compressed latent vectors, both in terms of locations and energy values. In addition, Figure 9.6 shows the EMD values corresponding for 4800 reconstructed jets. Relatively low values of the EMD imply a high-level of similarity between GVAE reconstructed and fully-simulated jets. Testing the GVAE with mincut pooling gives us a RMSE value of 1.7378. In terms of inference time, our GVAE model spends a total of 0.0638 seconds on a batch of 64 jets, which is orders of magnitude smaller than running full simulation.

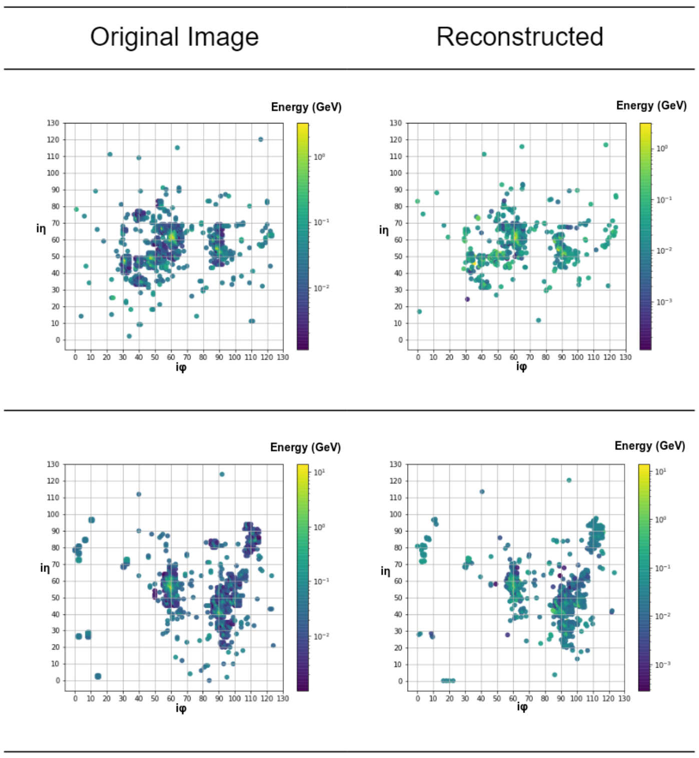


Figure 9.5: Reconstructed GVAE jet (right) in the detector as compared to the real simulated jet (left)

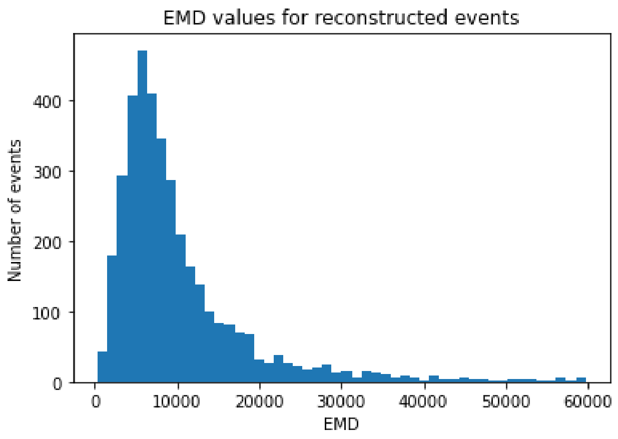


Figure 9.6: Earth Mover Distance values histogram

### 9.6.2 Graph generative model with edgepooling

We compare the graph architecture above with a graph autoencoder that takes performs EdgePooling to compress that graph representation of the jets under study. The decoder uses the UnPooling layer suggested by [73]. Results show the inability of the EdgePooling graph model to accurately learn the jet representations and reconstruct their topology. The RMSE in testing is 19.03258 which is 10 times higher than the RMSE value for the proposed model. On the one hand, Figure 9.7 show the reconstructions provided by this model displaying mediocre topology reconstructions and energy scales which are biased towards the highest values. On the other hand, the EMD plot corresponding to EdgePool-based GVAE shows that the EMD values are around 10 times higher than those of the model suggested above. Finally, the inference time presented by EdgePooling for the same batch of 64 samples is 2.1587 seconds as compared to 0.1235 seconds with GVAE. This is mainly due to the slow nature of the EdgePooling operations. The latter required further optimization in the process of aggregating node features in the hidden space and finalizing the edge score for graph contraction.

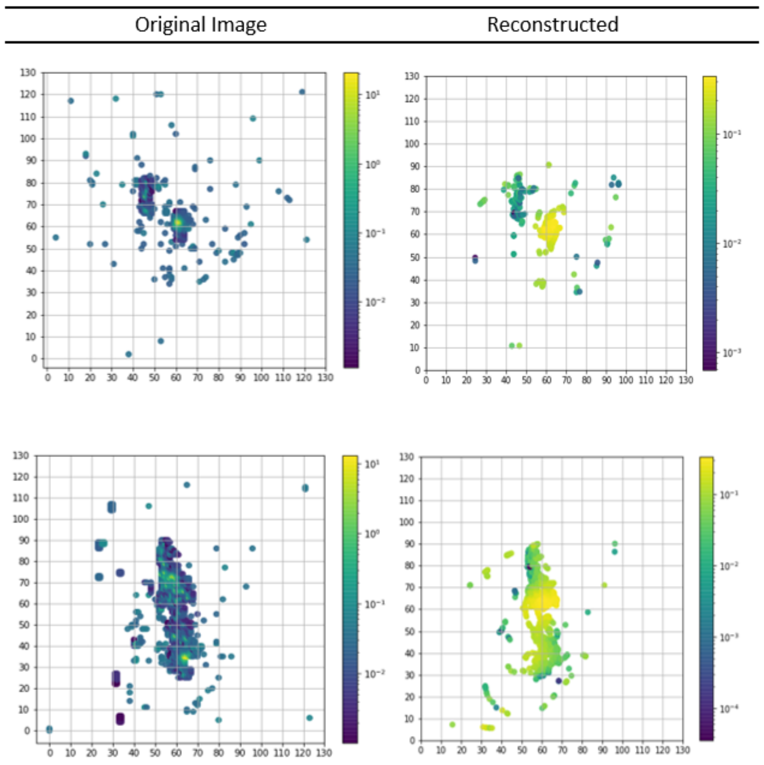


Figure 9.7: Reconstructed GVAE jet (right) in the detector as compared to the real simulated jet (left) using EdgePooling

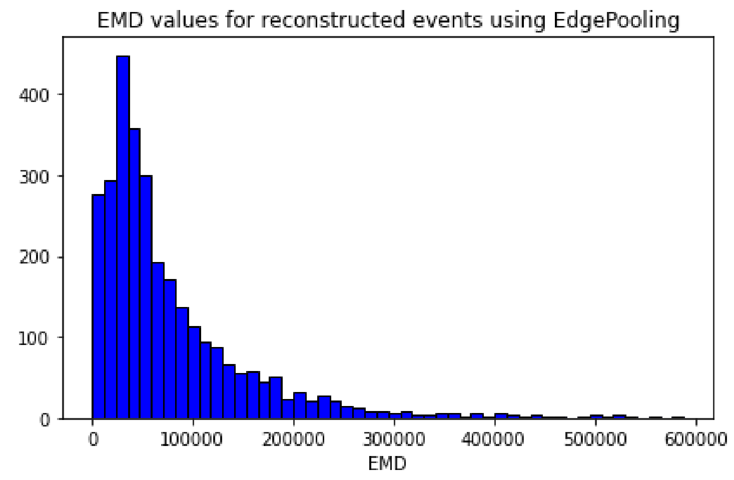


Figure 9.8: Earth Mover Distance values histogram for EdgePooling

### 9.6.3 Regular AE

The figure below shows reconstructions of jets through a regular autoencoder. It is notable that the autoencoder model is not able to accurately learn the representations of the jet events as compared to a graph autoencoder. Given that this approach is image-based, the EMD metric cannot be applied to it as the latter is solely for point cloud data. That being said, one additional advantage of convolving directly on point clouds and the usage of EMD on them is the permutation invariant property of this metric with regards to non-Euclidean data. Table 9.1 shows a comparison of the number of model parameters for each of our graph model, the graph EdgePooling model and a regular autoencoder, respectively.

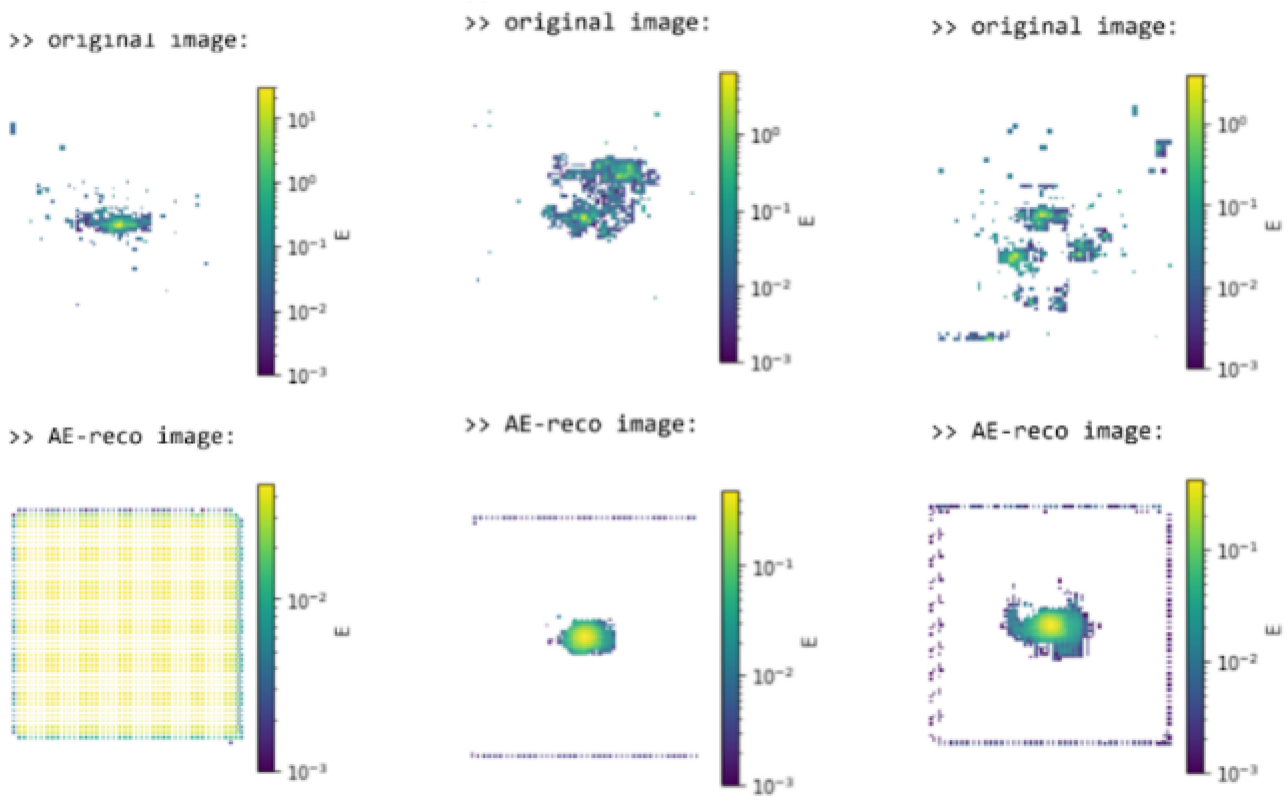


Figure 9.9: Regular Autoencoder reconstruction results

| Model | Mincut Graph VAE | EdgePool GVAE | AE |
|---|---|---|---|
| Number of parameters | 104 511 | 118 870 | 3 093 810 |
| Inference time (s) | 0.0638 | 2.1587 | 0.1478 |

Table 9.1: Model parameter comparison

## 9.7 Computational complexity

### 9.7.1 Regular VAE Complexity:

For a regular VAE, we proceed by performing regular convolution operations given as follows (See Section 4.8):

$$G[m,n] = (f * h)[m,n] = \sum_j \sum_k h[i,j] f[m-i, n-j] \tag{9.7}$$

where m and n represent the row and column indices, respectively. From the equation above, we realize that the number of operations is proportional to the filter size on the one hand and the image's row and column dimensions on the other hand, bearing in mind that the stride and padding effects are minimal with respect to the latter variables i.e for either stride=1 or 2 the compute term is still dominated by the row/column dims. In addition, we account for the feature dimension $d$ as it indicates how many channels of arrays we are dealing with. Consequently, a single layer convolving over one array has a compute term proportionality given by:

$$\mathcal{O} \propto s \times s_2 \times w \times h \times d \tag{9.8}$$

Generally speaking the filter size is square hence $s2 = s$ leading to

$$\mathcal{O} \propto s^2 \times w \times h \times d \tag{9.9}$$

At this stage, we extend our model to accommodate multiple layers with pooling. By convention, we proceed with an assumption that the channel upscaling within the encoder has a factor of 2, and that the image size is reduced to half by dimensionality reduction/pooling. To study the effect of performing multi-layer pooling and convolution to double the colour dimensions, we proceed as follows: Let the right term of equation 8.7 equal a constant $\alpha$.

**Given:**

- $out\_imsize = \frac{1}{2} \times in\_imsize = \frac{1}{2} \times size_{height} \times \frac{1}{2} \times size_{width}$
  $= \frac{1}{4} \times size_{height} \times size_{width} = \frac{1}{4} \times w \times h$
- $out\_channels_{encoder} = 2 \times in\_channels_{encoder}$

Therefore, we model a multi-layer encoder with convolutional and pooling operations as a series S given by:

$$S = \alpha + \frac{1}{4} \times \alpha + \frac{1}{4} \times \frac{1}{4} \times \alpha + \ldots + (\frac{1}{4})^l \times \alpha = \sum_{l=0}^{L} \alpha(\frac{1}{4})^l = \alpha \sum_{l=0}^{L} (\frac{1}{4})^l \tag{9.10}$$

At this stage, we want to find the upper bound of the equation S as a function of $\alpha$. In complexity theory, this upper bound indicates the maximum growth rate that a certain function/model can achieve, hence providing us with a description of the asymptotic behavior of the model's compute complexity. We say that our model (S in this case) grows at the order of this upper bound. Having said that, we notice that S in equation 8.8 is a geometric series in analogy to $S_n \equiv \sum_{k=0}^{n} r^k = \frac{1-r^{n+1}}{1-r}$. Since $-1 < r < 1$ in our case (r=0.25), the convergence at $n \to \infty$ becomes: $S \equiv S_\infty = \sum_{k=0}^{\infty} r^k = \frac{1}{1-r}$. As a result, performing infinite such layers in the model's encoder will be up-bounded as:

$$\alpha \sum_{l=0}^{L} (\frac{1}{4})^l \leq \alpha \sum_{l=0}^{\infty} (\frac{1}{4})^l \tag{9.11}$$

$$= \alpha \frac{1}{1-\frac{1}{4}} = \frac{4}{3}\alpha \tag{9.12}$$

Therefore, performing infinite convolutional layers within a variational autoencoder with the first given assumption will converge towards the result obtained in Eq 8.10. In other terms, the encoder model is bounded by $\alpha = s^2 \times w \times h \times d$. At this stage, we embed the second assumptions for downsampling using Eq 8.9 by considering that the colour channels will double in size within the encoder layers, eg: (64,128,256,etc ...). This results in the following equation:

$$s^2 \times w \times h \times d \sum_{l=0}^{L} (\frac{1}{4})^l \times 2^l \leq s^2 \times w \times h \times d \sum_{l=0}^{\infty} (\frac{1}{4})^l \times 2^l \tag{9.13}$$

$$\Rightarrow s^2 \times w \times h \times d \sum_{l=0}^{L} (\frac{1}{2})^l \leq s^2 \times w \times h \times d \sum_{l=0}^{\infty} (\frac{1}{2})^l \tag{9.14}$$

Given another geometric series with $r = \frac{1}{2}$, we get:

$$\Rightarrow \leq s^2 \times w \times h \times d \sum_{l=0}^{\infty} (\frac{1}{1-\frac{1}{2}}) = 2 \times s^2 \times w \times h \times d \tag{9.15}$$

Finally, since the entire baseline model is an encoder-decoder, and knowing that the decoder has the same architecture as the encoder in reverse, we just multiply the above equation by 2 and obtain the final upperbound UB in terms of a given batch size to be:

$$\boldsymbol{UB = 4s^2whd \times batch\_size} \tag{9.16}$$

### 9.7.2 GVAE Complexity:

The complexity of our model is mostly centered towards the GraphSAGE layers. As discussed in Section 9.2, GraphSAGE operates by sampling a set of random k nodes from L-hop neighbourhoods in order to reduce bias with regards to neighbouring nodes on the one hand and aggregate from as further as possible from the central node on the other hand. The complexity of a GraphSAGE layer is hence shaped by $k^L$ for the convolution operation on a single node. For a whole batch $b_s$ of graphs with $n_G$ nodes each having feature dimension $d$, the complexity is proportional to $b_s n_G k^L d^2$. For L layers of GraphSAGE present in both our encoder and decoder, the complexity is proportional to the term given by

$$\mathcal{O} \propto 2Lb_s n_G k^h d^2 + kn_G D \tag{9.17}$$

**How does the the function above converge?**
Equation 8.5 is dominated by the first term, for which we start by investigating its

convergence when $L \to \infty$. Let variable $\alpha = n_G \times k^h \times d^2$ describe the dominant components of equation 9.17. Assuming infinite graph convolutional layers, the equation converges as follows:

$$S = \alpha + \frac{1}{2} \times \alpha + \frac{1}{2} \times \frac{1}{2} \times \alpha + \ldots + (\frac{1}{2})^l \times \alpha \tag{9.18}$$

which, similar to the VAE case above, is a geometric series having $r = \frac{1}{2}$. Hence:

$$S = \sum_{l=0}^{L} \alpha(\frac{1}{2})^l \leq \sum_{l=0}^{\infty} \alpha(\frac{1}{2})^l \tag{9.19}$$

At this point, we use the second given bullet for the graph model case stating that colour channels will double in size for each layer. This leads to

$$S \leq \alpha \sum_{l=0}^{\infty} (\frac{1}{2})^l \times 2^l \approx \alpha \tag{9.20}$$

Finally, accounting for the decoder, the function S is bounded by 2. Therefore, the upper bound for our graph model in terms of a given batch size is:

$$\boldsymbol{UB = 2\alpha = 2n_G k^h d^2 \times batch\_size} \tag{9.21}$$

### 9.7.3 Comparison Tables

Since we have two different models with different terms, the most valid way for comparison is by tabulating the most common values for these variable terms and comparing them with respect to the batch size. For instance, a graph model does not have a kernel filter of size s, nor a grid input of size w by h. Similarly, a regular autoencoder is not defined by a specific number of nodes and L-hop neighborhood. That being said, the first table shows the most common results for the filter size, width and height represented by s, w and h respectively. The output for that table is then given by $s^2wh$. The second table shows the different values for $n_G$, k and h referring to the number of nodes, the number of neighbours and hop-neighborhood, respectively. Given that our jets do not go beyond 1000 hit on most of the time, we test for 1000 on the one hand, and take a worst case scenario of 1500 hits on the other hand. Then, the output is given by $n_G k^h$. The outputs for the tables belonging to the Graph model are clearly much lower as compared to the AE model.

### 9.7.4 Comparison with traditional Monte Carlo

For benchmarking, we compare the inference time of that of the conventional methods, such as Monte Carlo simulations, that are used for approximating the

| s | w | h | Output |
|---|---|---|---|
| 3 | 125 | 125 | **140625** |
| 4 | 200 | 200 | **640000** |
| 2 | 75 | 75 | **22500** |

Table 9.2: Autoencoder average variable values

| $n_G$ | k | h | Output |
|---|---|---|---|
| 1000 | 4 | 2 | **16000** |
| 1000 | 8 | 2 | **64000** |
| 1500 | 4 | 1 | **6000** |

Table 9.3: Graph Autoencoder average variable values

probability density function. Such method takes 45 seconds for the event batch of the same size. In contrast to this results, the graph method takes around 0.1 second for the inference, which is over 400% speedup.

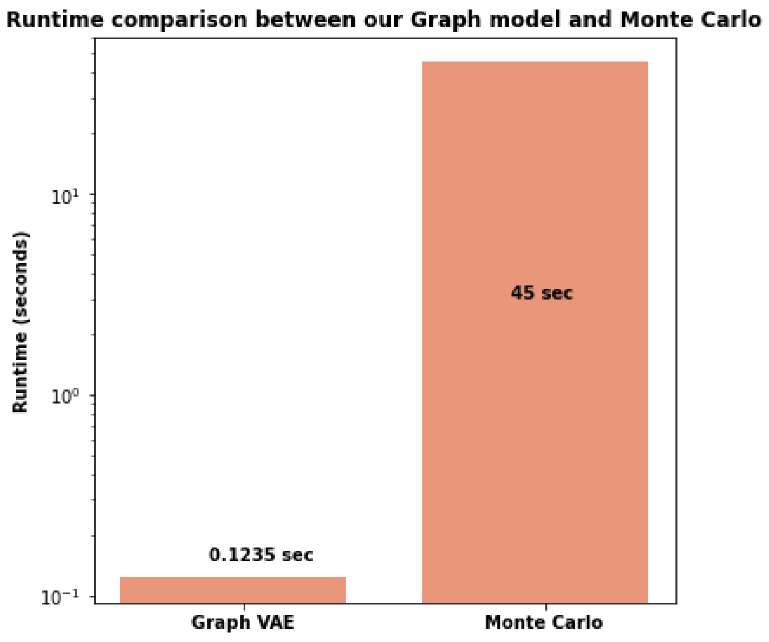


Figure 9.10: Speedup comparison for boosted top quark jets

## 9.8 Discussion

We therefore conclude that our approach is successful in reproducing particle physics boosted jets data at high-levels of fidelity and with acceptably low inference times relative to other models. For instance, it is clear from the figures that the model spans the range of the real jet distributions accurately in terms of topology on the one hand and energy scale on the other hand. In addition, we quantify the difference between real and reconstructed point cloud distributions by means of the Earth Mover Distance metric. A comparison of the mincut pooling approach and the EdgePooling one shows that the former provides a better range for the work required to shape its model output distribution into the real one. Finally, the RMSE achieved by the graph generative model with mincut pooling is the lowest compared to GVAE with EdgePooling and the regular VAE.These results pave the way for more further investigations to be done on sparse detector data by means of graph representations.

For future work, the scope of this project could safely be extended for applications that include multiple channels such as the Tracker system and the Hadronic Calorimeter (HCAL). Furthermore, the promising results shown by graph models on reconstruction tasks make them a suitable candidate for other applications such as jet tagging and particle tracking. While some studies have already started with this approach such as ParticleNet [5], the added value of hierarchical learning by means of novel graph pooling layers remains under-explored. Finally, given that graph networks can accurately reconstruct jet events as point clouds in a timely efficient manner, we are then interested in seeing how our model scales on multiple GPUs for even faster computations to occur. Having said that, this scaling process is discussed in the next chapter.

# Chapter 10

# FALCON: GPU Acceleration on NVIDIA GPU

## 10.1 Overview

This year our FALCON project was accepted to the 2020 Helmholtz GPU Hackathon, a five-day event organized by Helmholtz-Zentrum Dresden-Rossdendorf (HZDR), Jülich Supercomputing Centre (JSC) and Helmholtz Federated IT Services Software Cluster (HIFIS) [103]. The Hackathon's duration was of 5 days during groups of scientists interested in computing and porting their applications to GPU have gained a hands-on experience in GPU acceleration and code optimization under the guidance of experts from NVIDIA and other institutes and industries.
To proceed, the participating groups were given two options for accessing GPU supercomputer facilities:

- The first option is to connect to the TAURUS cluster present at TU Dresden in Germany. This system is composed of the following hardware: a total of 128 NVDIA K80 GPUs in 64 nodes and Intel Haswell CPUs in addition to NVIDIA V100 GPUs with IBM POWER9 CPUs.
- The second options is using raplab-hackathon which contains 10 NVIDIA DGX-1 compute nodes with 8 NVIDIA Volta V100 GPUs per node. Each GPU has a 16 GB RAM, which makes it easier to train larger batches and models. Finally, the cluster has 48 core Intel (R) Xeon (R) Gold 5118 CPUs per node with 192 GB of RAM @2.30 GHz.

## 10.2 Profiling FALCON's GVAE model

Prior to fully training our model and porting it the multi-GPUs, we need to scrutinize our code to investigate the data loading efficiency and any potential

processes, especially CPU-based, that could be affecting the training speed. For this purpose, we apply a profiling run on our code for a few training iterations (not epochs) using the NVIDIA Nsight Systems's Visual Profiler. The profiling process can be visualized in Appendix C. It was notable that our code was spending too much time on the DataLoading process, a CPU-based task. In fact, prior to the hackathon, we adopted the strategy of creating graphs from the raw detector data and storing them as ".pt" files containing the information about node features and edge indices for each sample. This method had resulted in a 2 GB file containing Data objects with dictionaries 'X' and 'edges' referring to the feature and adjacency matrices, respectively. Consequently, moving the entire training dataset from the CPU to the GPU was computationally intensive even when increasing the number of workloaders.
To tackle this issue, we decided to generate the graphs representing the jets on the spot prior to sending them to GPU for training. In fact, we stored our raw data in parquets files which allow suitable "lazy-loading" of the data, a very useful option in our case where the data is too large to be loaded entirely at once. In other words, our new dataloading strategy is as follows:

- Generate a batch of the raw detector data on the spot.
- Sample the non-zero hits' x,y locations.
- Find the energy values at those locations and concatenate them with the corresponding x,y coordinates.
- Generate a graph using the k-nearest neighbour algorithm resulting in a feature matrix X and adjacency matrix A for each sample in the batch.
- Save the samples as dense data objects and send them to the GPU for training

Having said that, adopting the new loading strategy resulted in a 50% reduction in training time!

## 10.3 Distributed Deep Learning

Having profiled our model on trained it for performance monitoring using the new dataloading strategy on a single GPU, we were able to proceed with porting our model to check how it scales on multiple GPUs. Multi-GPU programming is necessary on many occasions where scientists are either dealing with datasets too large to fit into single GPU memory or developing models with a large number of trainable parameters. These obstacles apply to our case as we are training thousands of graphs each of which contains an average of 700 nodes and 4 times the number of edges. Having said that, it is in our best interest to distribute our training on multiple GPUs. To proceed, there are two options under disposal for GPU acceleration: data parallelization and model parallelization. The former approach is crucial in scenarios where the dataset takes much of the memory space (or is even bigger than the allocated memory) which prompts the user to send smaller batches hence increasing training time. In that case, multiple batches can be sent to different GPUs where the same model layers operate on them (Multiple Input Single Instruction) after which the GPU nodes synchronize and aggregate the output layers for backpropagation to take place. On the other hand, model parallelization consists of distributing the deep learning model parameters into multiple devices operating sequentially on a single batch of data. At that point, every time a batch is transformed through the layers on one GPU device and sent to the next one, another batch is simultaneously allocated to the first GPU and so on. Both methods play a crucial role in accelerating training and inference of DL models. Yet, data parallelization is much less complex to apply and is usually the first option for data scientists to consider.At this point, we proceed with scaling our model to multiple GPUs using Horovod [104], a deep learning library for distributed training of DL models supporting many frameworks such as PyTorch, TensorFlow and MXNet. Horovod operates based on MPI (Message Passing Interface) concepts that dictate our mutlti-gpu process. For instance, Horovod takes as input the *size* parameter referring to the number of processes P in training, followed by the *Rank* parameter unique to process and numerated from 0 to P-1. Finally, *LocalRank* indicates the unique process ID within each device (0 to $N_{GPUS}$).

## 10.4 Tesla V100 Architecture

Amid the rising significance of High Performance Computing (HPC) on the one hand and the surge in data availability on the other hand, it was crucial to combine the aspects of HPC and AI for advanced scientific research to take effect. For instance, the extension of AI to HPC applications could pave the way for large scale simulations for astrophysics, molecular dynamics, climate science and other scientific fields. Having said that, NVIDIA's 2017 GTC (GPU Technology Conference) saw the introduction of the Tesla V100 GPU, currently one of the world's most powerful GPU devices providing a platform for accelerated Deep Learning research, HPC and other graphics applications [105]. On the other hand, this processor belongs to the Volta GPU architecture family which played a massive role in shaping the success of Artificial Intelligence-based applications. In fact, the Volta architecture is characterized by the embedding of CUDA cores and Tensor Cores which highly enhance compute performance for floating point operations [106]. CUDA cores are specialized compute hardware that perform parallel processing. In contrast to CPU cores, CUDA cores are much numerous ranging from hundreds to thousands of cores. Having said that, the Tesla V100 architecture contains 5120 CUDA cores spread among Streaming Multiprocessors (SM). On the other hand, the Volta architecture provided the first generation of Tensor cores which have notably enhanced the speedup of deep learning architectures. A tensor core is a compute unit that specializes in matrix multiplication and is characterized by its ability of downscaling the entries to FP16 for faster computation to take place and returning either an FP16 or an FP32 matrix as output. Accordingly, a user would choose an optimum between gained speedup and lost precision for their GPU-based application.

## 10.5 Scaling FALCON on multiple GPUs

We train our FALCON model on the raplab-hackathon cluster using Volta V100 GPUs having 16 GB of RAM. Following profiling and code optimization for enhance CPU performance, we scale the training on multiple GPUs using the Horovod library. To get a baseline performance, we compare the results to the training done on a single GPU with a batch size of 32 for 100 iterations i.e a total of 3200 graph samples. The performance of the scaling is documented both in the section below.

## 10.6 Results

Throughout scaling we notice an increase in the performance with an increase in the GPU devices used. To calculate the resulting speedup from scaling, we take as reference the Mean Execution Time (MET) resulting from training on one GPU. For $N_{GPUs} = 2$, the mean execution time is 85.48 seconds. Bearing in mind that we are training twice the amount of data overall, we get MET per GPU = 42.74 seconds. The resulting speedup for $N_{GPUs} = 2$ is given by

$$\frac{MET_{singleGPU}}{METperGPUusingN_{GPUs}} = \frac{69.34}{42.74} = 1.62 \tag{10.1}$$

Applying the same speedup calculation method, going from 2 to 4 GPUs, we get speedups of 1.62, 2.19 and 2.73, respectively. Therefore, we conclude that our model scales well on multiple GPUs using Horovod, which is a useful information for future work on FALCON and similar graph generative models. Nevertheless, we have been faced with some minor downsides. For instance, Table 10.1 shows some loss in performance due to parallelization efficiency. This could be explained by an inefficient communication between the GPUs once parallel batch training has been done, hence increasing latency which in turns affects the training time. This issue can be overcome at a lower level where the user proceeds with programming the kernel directly, which can be done using CUDA programming. This task is out of the scope of this thesis for now as we are solely interested to get a general picture of how the model scales to multiple GPUs. Further optimization could be deferred to future work.

On the other hand, we used NVIDIA's Automatic Mixed Precision (AMP) library which enables mixed precision training of deep learning models with support to distributed parallel training. This approach has the potential to boost the speedup of model training by enabling the support of FP16 operations throughout the model's layers. This step offers several benefits such as the reduction in required memory for FP16 as compared to 32-bit floating points. In addition, the data transfer and linear algebra operations are performed much faster using lower precision formats.

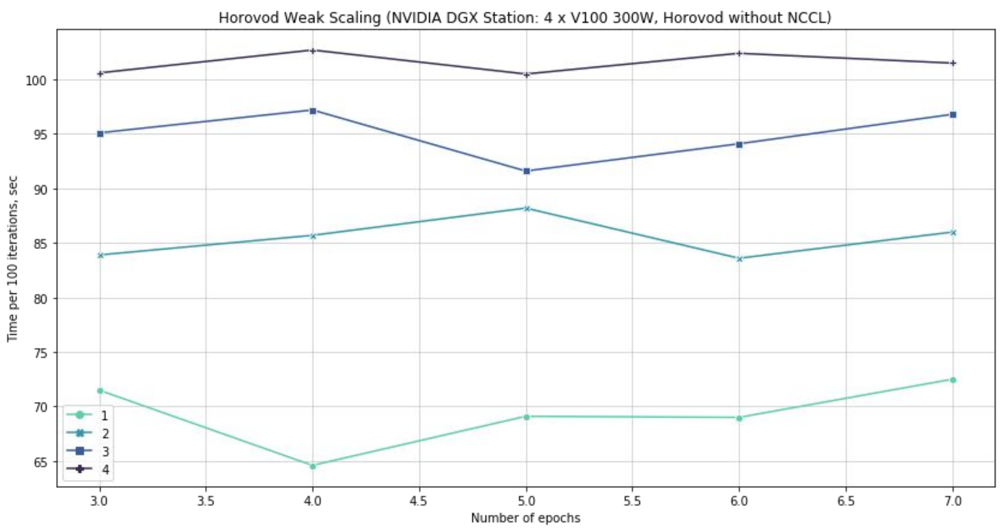
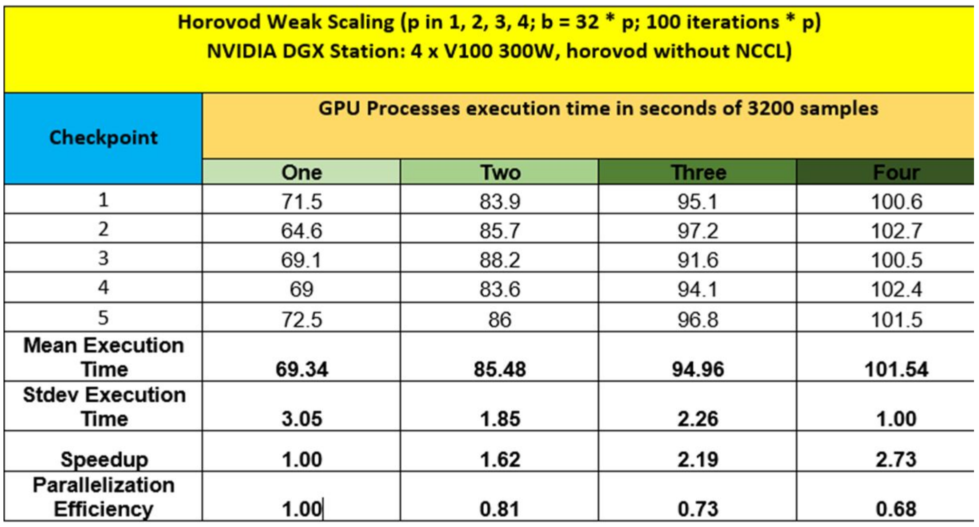


Figure 10.1: Plot of multi-gpu performance



**Horovod Weak Scaling (p in 1, 2, 3, 4; b = 32 * p; 100 iterations * p)**
**NVIDIA DGX Station: 4 x V100 300W, horovod without NCCL)**

| **Checkpoint** | **GPU Processes execution time in seconds of 3200 samples** | | | |
|---|---|---|---|---|
| | **One** | **Two** | **Three** | **Four** |
| 1 | 71.5 | 83.9 | 95.1 | 100.6 |
| 2 | 64.6 | 85.7 | 97.2 | 102.7 |
| 3 | 69.1 | 88.2 | 91.6 | 100.5 |
| 4 | 69 | 83.6 | 94.1 | 102.4 |
| 5 | 72.5 | 86 | 96.8 | 101.5 |
| **Mean Execution Time** | **69.34** | **85.48** | **94.96** | **101.54** |
| **Stdev Execution Time** | **3.05** | **1.85** | **2.26** | **1.00** |
| **Speedup** | **1.00** | **1.62** | **2.19** | **2.73** |
| **Parallelization Efficiency** | **1.00** | **0.81** | **0.73** | **0.68** |

Table 10.1: Table of multi-gpu performance

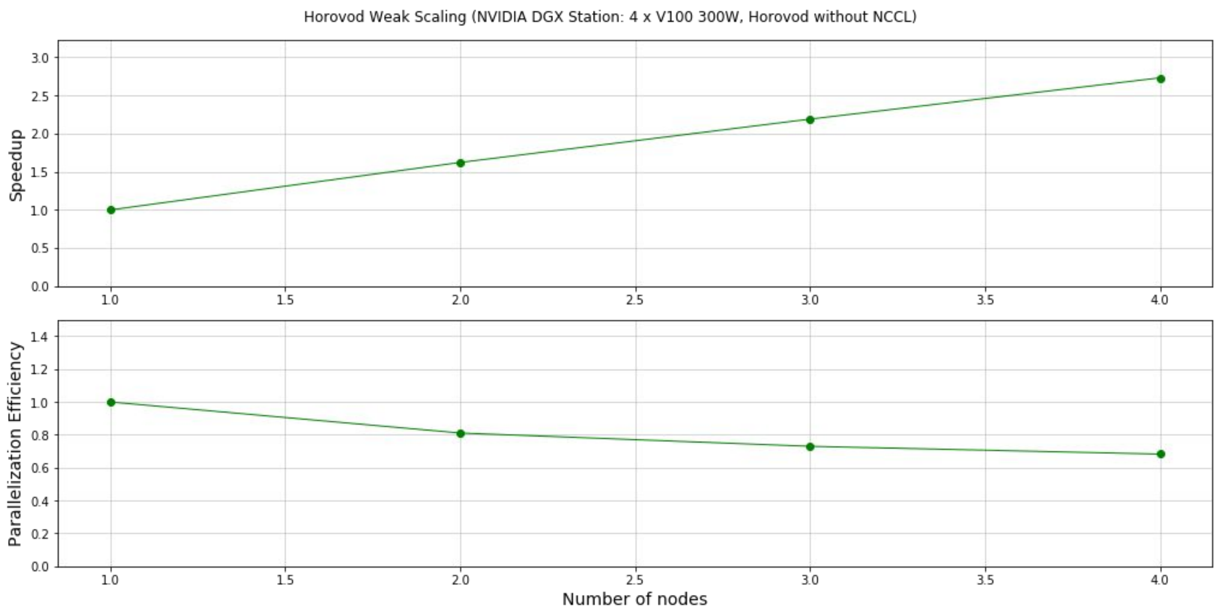


Figure 10.2: Comparison plot of how our model scales to multiple GPUs

## 10.7 Discussion

Throughout this event we were able to perform further optimization of our code for jet reconstruction through graph autoencoding. Our first step was to profile the code to monitor the process load taking place in the CPUs and GPUs. As the dataloading was timely expensive even with an increased number of threads per process, we decide to adopt a different strategy for batch loading to remove this bottleneck. Consequently, we generate batches of graphs on the spot in the CPU prior to sending them to the training model on the GPU. This change has reduced the training time by 50%, allowing us to proceed with scaling on multiple GPUs. Having trained on a single Tesla V100 GPU as a baseline, we proceeded by scaling the model to multiple GPUs using the Horovod library. Results show a notable speedup as we distribute to multiple devices going from 2 to 4. Yet, some performance is lost in the parallelization due to some inefficient communication between the GPUs following weight propagation process. For future work, users are necouraged to try programming the MPI kernel for further optimized parallelization performance which would boost the speedup even further. Finally, the apex framework was used for distributed training while enabling mixed precision training of our graphs. No significant change in the training time was noted with the usage of FP16 throughout the layers. This could be due to the incompatibility of the graph convolution operations provided by dependencies with the apex commands. Another possible reason is the dense nature of our batches. In the future, it is recommended to sparsify the input graphs and their corresponding

convolutional layers accordingly.
In a nutshell, we can confidently report that graph-based architectures hold a promising potential in high energy physics applications. In contrast to image-based approaches, graph neural networks allow us to operate on a range of hundreds of hits as compared to thousands and even more in alternative ways.

# Chapter 11

# Muon momentum estimation in the CMS detector

We aim to explore to potential offered by graph networks in LHC physics tasks. In our case, we try to infer muon momentum from data collected in the Cathode Strip Chambers.

## 11.1 Dataset

Our dataset consists of more than 3 million muon events generated using Pythia. The dataset is an npz file containing 2 numpy arrays ‘variables’ and ‘parameters’. The first array ‘variables’ contains 87 features, 84 of which serve as input variables and 3 other serving as road variables: pattern straightness, zone, median theta, respectively. This is due to the fact that we have 12 detector planes in the CSCs, each of which contains 7 features (12*7=84). For a better perspective on the data format of this array please refer to tables in Appendix D. The second array ‘parameters’ contains 3 columns described in Table 4.
During pre-processing, we remove samples with number of hits smaller than 4, indicating noisy data whose output cannot be determined. Consequently, the number of samples has been reduced to 1.4 million events in order to account for the events recorded in all muon chambers. The input consists of hit coordinates of the muons at four different chambers of the CSC, while we consider the muon momentum for the output. At this stage,it is worth noting that our data is highly imbalanced and biased towards muon velocities with low to medium momenta. Therefore, we consider scaling techniques for highly skewed distributions such as log transforms and inverse transforms. In our case, applying the latter resulted in the most ”Gaussian-like” distributions as shown in Fig 11.1 below. Hence, we proceed with predicting the inverse momenta of muons.

So far, three baseline models have been tried:a LightGBM model, a fully connected network and a convolutional neural network (CNN). All three architectures

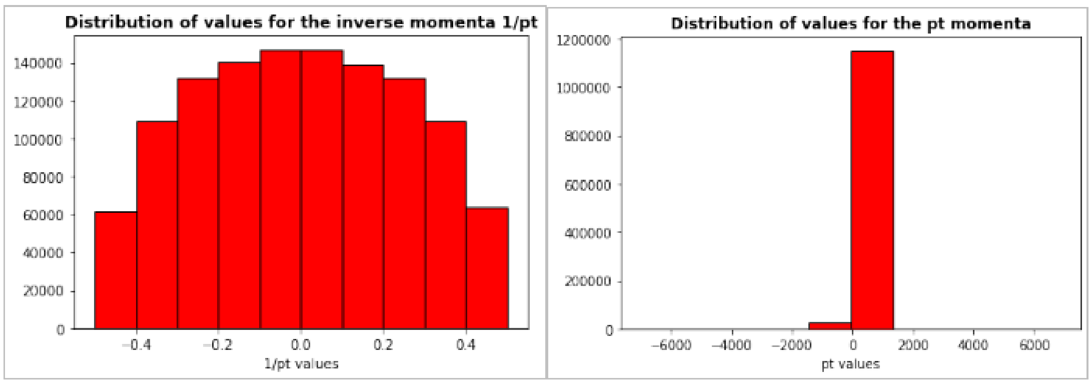


Figure 11.1: Distribution of pt momentum (right) as compared to 1/pt momentum (left)

will be trained on predicting $1/pt$ and will be compared to the performance of a Graph Neural Network.

## 11.2 Methodology

We suggest Graph Neural Networks (GNN) as a new method to measure momenta in the muon chambers. If successful, this family of architectures could pave the way for new deep learning applications in particle physics regarding the trigger system. To proceed, we investigate model performance in 3 ways: regression on the muon's $p_T$ value, regression on the muon's inverse $p_T$ value given the better distribution it provides (Fig 11.1) and classification of 4 classes of ranges within which the momenta fall. We implement three architectures that serve as baseline for comparison.

- Our first baseline is a Fully-Connected Neural Network (FCNN) composed of 5 dense layers of 512,256 and 3x128 neurons, respectively. The ReLU activation is used following each linear transformation. In addition, for the cases of $p_T$,$1/p_t$ and 4 classes, the output functions are linear, sigmoid and softmax, respectively.
- Our second baseline is a Convolutional Neural Network (CNN) characterized by 3 convolutional layers each having a kernel size of 2x2 and channels of 512,128 and 128, respectively. Following the convolutional block is a fully connected network that takes as input a flattened CNN output and transforms it through 4 fully-connected layers (256,128,128,64) to get an eventual output.
- Our final baseline is a LightGBM model, a decision tree-based gradient boosting framework. We perform bayesian optimization on a set of param-

eters to further improve our training. The optimized parameters are the learning rate, bagging fraction, minimum child sample, maximum depth, Lambda 1, Lambda 2, feature fraction and early stopping. Their corresponding values can be found in Appendix D.

Our main model is a GNN that performs a set of spatial convolution operations in addition to attention layers. A diagram describing our model can be found in Figure 11.2, whereas GCN refers to a Graph Convolution block with the corresponding input and output channels marked within. As seen in the diagram, our graph input undergoes 4 spatial convolutions in total amid which 2 global attention layers are applied whose concatenation is market by the $\bigoplus$ symbol. Next, a three sets of fully connected layers transform the hidden representation of the graph into one vectorized output resulting in the desired momentum value or class.

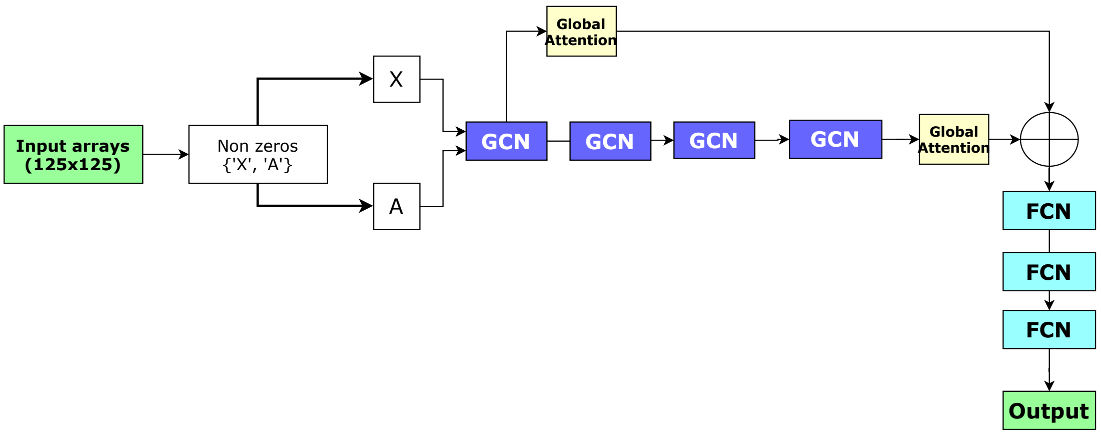


Figure 11.2: Graph Architecture for Muon Mmentum estimation: the raw data mapped into feature and adjacency matrix undergo a set of graph convolution operations followed by attention layers over two steps. The latter are concatenated and fed into a classification score through linear transformations

## 11.3 Results

Our GNN model shows improvements with respect to the baselines which include FCNN and CNN.The results are shown in the figures below for their predictions both on the regular momentum $p_T$ and the inverse momentum $1/p_T$. For instance, the GNN model results in lower Mean Absolute Error (MAE) in the regression tasks for both outputs. On the other hand, the bin classification performance is notably better with a graph-based architecture given that the f1-scores and accuracies are higher. We could deduce that for all models, high momentum particles are easier to detect as compared to low momentum ones. This is observed in both

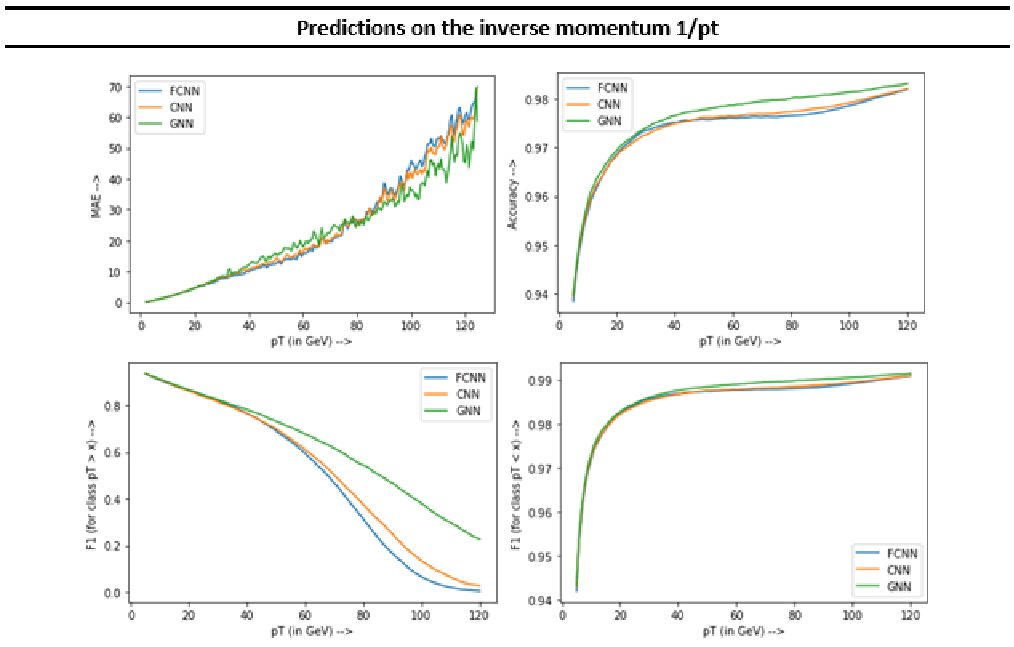


Figure 11.3

Figures 11.3 and 10.4 where we train on the momentum $p_T$ and inverse momentum $1/p_t$, respectively. For instance, the MAE values decrease with increasing $p_T$ ranges, while the bin classification accuracy increases, while the same behavior applies to the F1 scores.

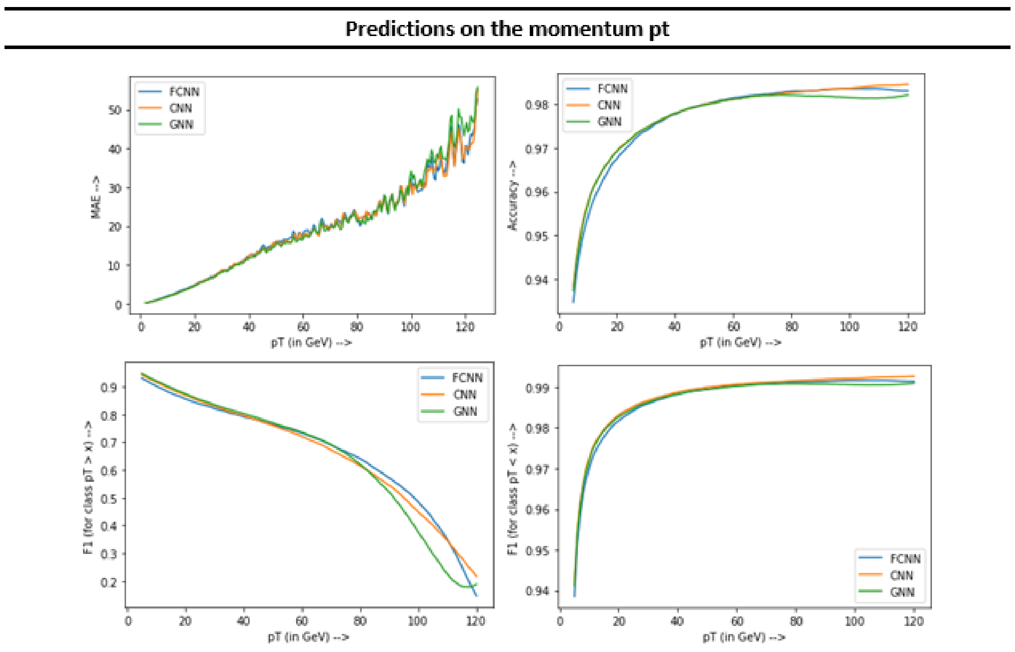


Figure 11.4

## 11.4 Discussion

We present a new perspective for measuring the momenta of subatomic particles in the muon chambers where we cast the muon data collected at the CMS detector into a geometric deep learning problem. We present a novel problem formulation that does not assume muon data to be Euclidean, i.e. grid-structured, but frames their data in a graph format. Advanced deep graph architectures such as graph convolutional networks with attention layers were investigated and were proven accurate and competitive with conventional approaches in estimating the regular or inverse momenta of muons. Provided the GNN accuracy, it would be interesting to investigate the efficiency of architectures such as message passing networks and interactions networks on a multitude of other regression tasks in particle physics.

# Chapter 12

# Conclusion

In this work we shed light on the potential of graph-based architectures for representing particle jets resulting from high-energy collisions on the one hand and subatomic particles in the muon chambers on the other hand. Graph neural networks tackle the issue of data sparsity in particle detectors by allowing the model to directly learn from the particle hits while disregarding empty cells during the training of the model. Our first application of graph architectures is in Fast Simulation. We develop a graph encoder-decoder model to learn the representation of Boosted Jets for future simulation purposes. Through spatial convolution, the model is able to learn the interactions between particle hits forming the topology.In addition, the GVAE model performs message passing based on the SAGE algorithms [99] in order to learn latent space representations of the point clouds as graphs. The latter is sequentially compressed by means of Mincut pooling to preserve the most representative nodes in latent space. This pooling layer learns a cluster assignment matrix through linear transformations, arranging the nodes with similar features into a unified neighborhood. Once encoding into latent space is done, a trained decoder with reverse encoder architecture upsamples the compressed vectors to the original reconstruction, leading to a proof-of-concept simulator. Our results show that our graph model can accurately learn to reconstruct point cloud events at the level of the Electromagnetic Calorimeter, as seen from the visual reconstructions and the MSE loss. On the one hand, comparing the Earth Mover Distance histograms of two Graph VAE models shows that the Mincut Pooling-based architecture performs better. On the other hand, the visual reconstructions obtained by applying a regular Autoencoder on Euclidean data shows the inability of such models to compete with the graph models when learning on sparse data. Eventually, a compute performance comparison shows lower numbers of model parameters for our Mincut Pooling GVAE model as compared to EdgePooling GVAE and regular AE. A complexity analysis with tabulated values of the most common parameters dominating the complexity equation shows that our model's computational efficiency is competitive with baseline approaches and could outperform them in many scenarios depending on the selected parameters. From a practical perspective, the inference time of our model has massively outperformed the EdgePooling-based model and shows competitiveness with a regular AE model as indicated in Table 9.1.

Next, we investigated the scalability of our model. Throughout the 2020 Helmholtz GPU Hackathon, we started on optimizing our code using NVIDIA Nsight Systems' Profiling Tools to check for any bottlenecks, especially in CPU-based processes. We proceed by changing the graph generation process within our code by implementing it on the spot after batch loading, resulting in 50% speedup in training. Using the Horovod library, we perform data parallel training proceeding from 2 to 4 Tesla V100 GPU devices resulting in speedups of 1.62, 2.19 and 2.73 respectively compared to a baseline of single GPU training.

Our second application of graph-based architectures is on muon momentum inference in Cathode Strip Chambers. Getting these measurements accurately is crucial for the trigger system operation as it allows better selection of muon batches which paves the way for scientists to study decay processes with higher reliability models. That being said, graph networks have been proven competitive with conventional machine learning and deep learning techniques in the estimation of muon momenta, with a momentum inference accuracy of at least 94% for low momentum samples until 98% for high momentum ones.

## Future Work

For future work, an extension of the Fast Simulation approach to multiple channels including the tracker and HCAL systems is recommended. One approach to proceed with this application is by making inter-detector connections of graphs and adding a detector class term to the loss function, resulting in an overall of 3 classes. Once such a model is ready, it could serve as a single GPU baseline to investigate the scalability of the model on multiple GPUs when simulating the entire 3 channels within the CMS detector. However, the mapping of multiple channels into point cloud graphs is not trivial and hence building its topology and applying graph pooling to its data remains an open question. In addition, the scalability of our model was hampered by the parallelization efficiency and hence future improvements include a re-programming of the kernel, whose improvement on the speedup cannot be estimated at the moment. Eventually, for the muon momentum inference problem, it is recommended for the current code and baselines to be run on FPGA, which will be the adopted hardware within the trigger system. The behavior of the model could deviate a little from the GPU's, therefore requiring some modifications.

## Broader Impact

The FALCON simulation work is part of an open-source project, and has a potential to impact many researchers who rely on particle simulations for physics studies. In terms of ethical and future societal consequences, the computational efficiency provided by the generative model presented in this work, allows to compete with computational and times of conventional Deep Learning techniques while outperforming their reconstruction accuracy. In the absence of adequate computational resources, this type of simulation would be an advantage to the research, as it provides an opportunity for a speedup of obtaining the results. At the same time, due to a need for some computational requirements of our implementation, those with very limited access to computing resources may be at a disadvantage. Finally, we set a platform for future simulation applications to take place, possibly covering more detector layers and wider ranges of information.

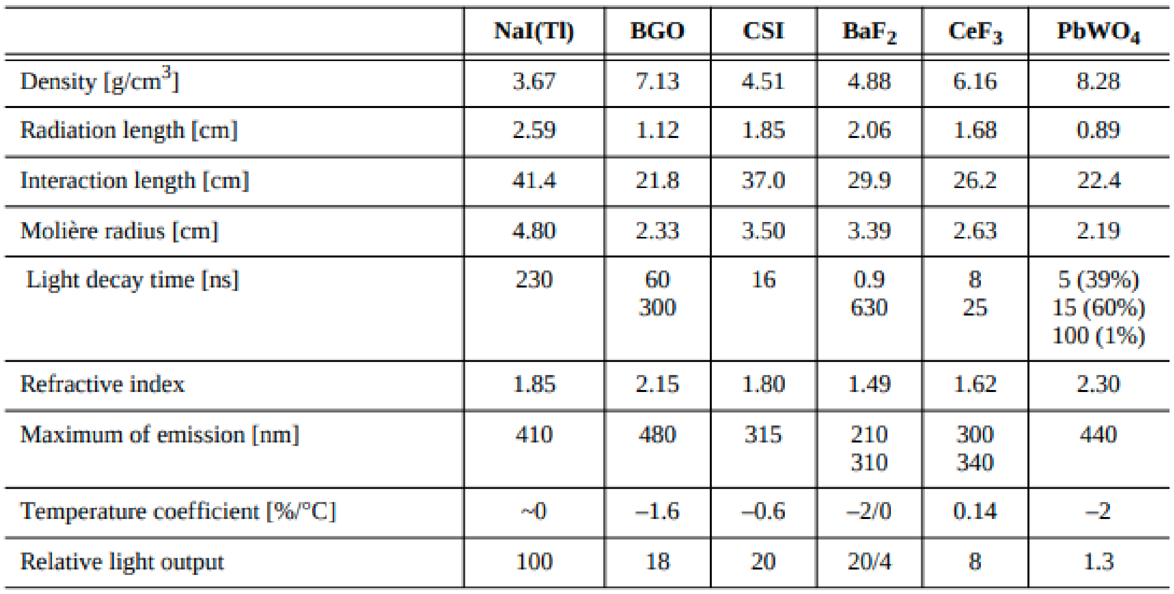
# Appendix A

# Abbreviations

| | |
|---|---|
| HEP | High Energy Physics |
| SM | Standard Model |
| LHC | Large Hadron Collider |
| CMS | Compact Muon Solenoid |
| CSC | Cathode Strip Chambers |
| ECAL | Electromagnetic Calorimeter |
| HCAL | Hadronic Calorimeter |
| QCD | Quantum Chromodynamics |
| ML | Machine Learning |
| BDT | Boosted Decision Tree |
| DL | Deep Learning |
| FCN | Fully-Connected Network |
| CNN | Convolutional Neural Network |
| RNN | Recurrent Neural Network |
| GDL | Geometric Deep Learning |
| GNN | Graph Neural Network |
| GCN | Graph Convolutional Network |
| PDF | Probability Distribution Function |
| VAE | Variational Autoencoder |
| GVAE | Graph Variational Autoencoder |
| GAN | Generative Adversarial Network |
| EMD | Earth Mover Distance |
| ROC | Receiver Operating Characteristic |
| AUC | Area Under the Curve |
| RMSE | Root Mean Square Error |
| MAE | Mean Absolute Error |
| GPU | Graphics Processing Unit |
| CUDA | Compute Unified Device Architecture |

# Appendix B

# Material comparison



| | NaI(Tl) | BGO | CSI | $BaF_2$ | $CeF_3$ | $PbWO_4$ |
|---|---|---|---|---|---|---|
| Density [$g/cm^3$] | 3.67 | 7.13 | 4.51 | 4.88 | 6.16 | 8.28 |
| Radiation length [cm] | 2.59 | 1.12 | 1.85 | 2.06 | 1.68 | 0.89 |
| Interaction length [cm] | 41.4 | 21.8 | 37.0 | 29.9 | 26.2 | 22.4 |
| Molière radius [cm] | 4.80 | 2.33 | 3.50 | 3.39 | 2.63 | 2.19 |
| Light decay time [ns] | 230 | 60<br>300 | 16 | 0.9<br>630 | 8<br>25 | 5 (39%)<br>15 (60%)<br>100 (1%) |
| Refractive index | 1.85 | 2.15 | 1.80 | 1.49 | 1.62 | 2.30 |
| Maximum of emission [nm] | 410 | 480 | 315 | 210<br>310 | 300<br>340 | 440 |
| Temperature coefficient [%/°C] | ~0 | –1.6 | –0.6 | –2/0 | 0.14 | –2 |
| Relative light output | 100 | 18 | 20 | 20/4 | 8 | 1.3 |

Comparison of different material candidates for ECAL sensors [107]

# Appendix C

# Porting Falcon to GPU

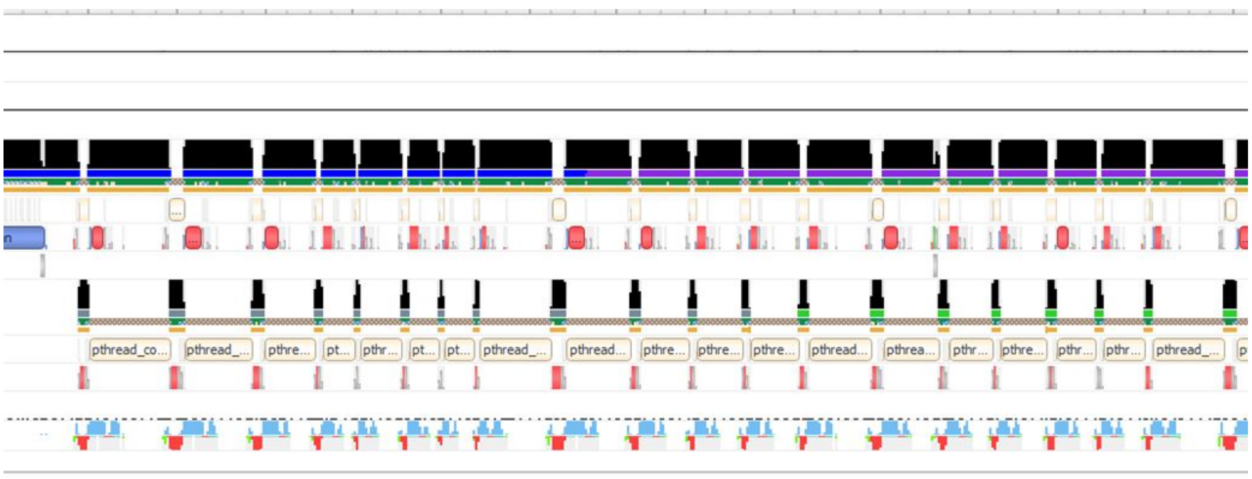


Figure: Profiling of FALCON's GVAE training using NVIDIA's Visual Profiler

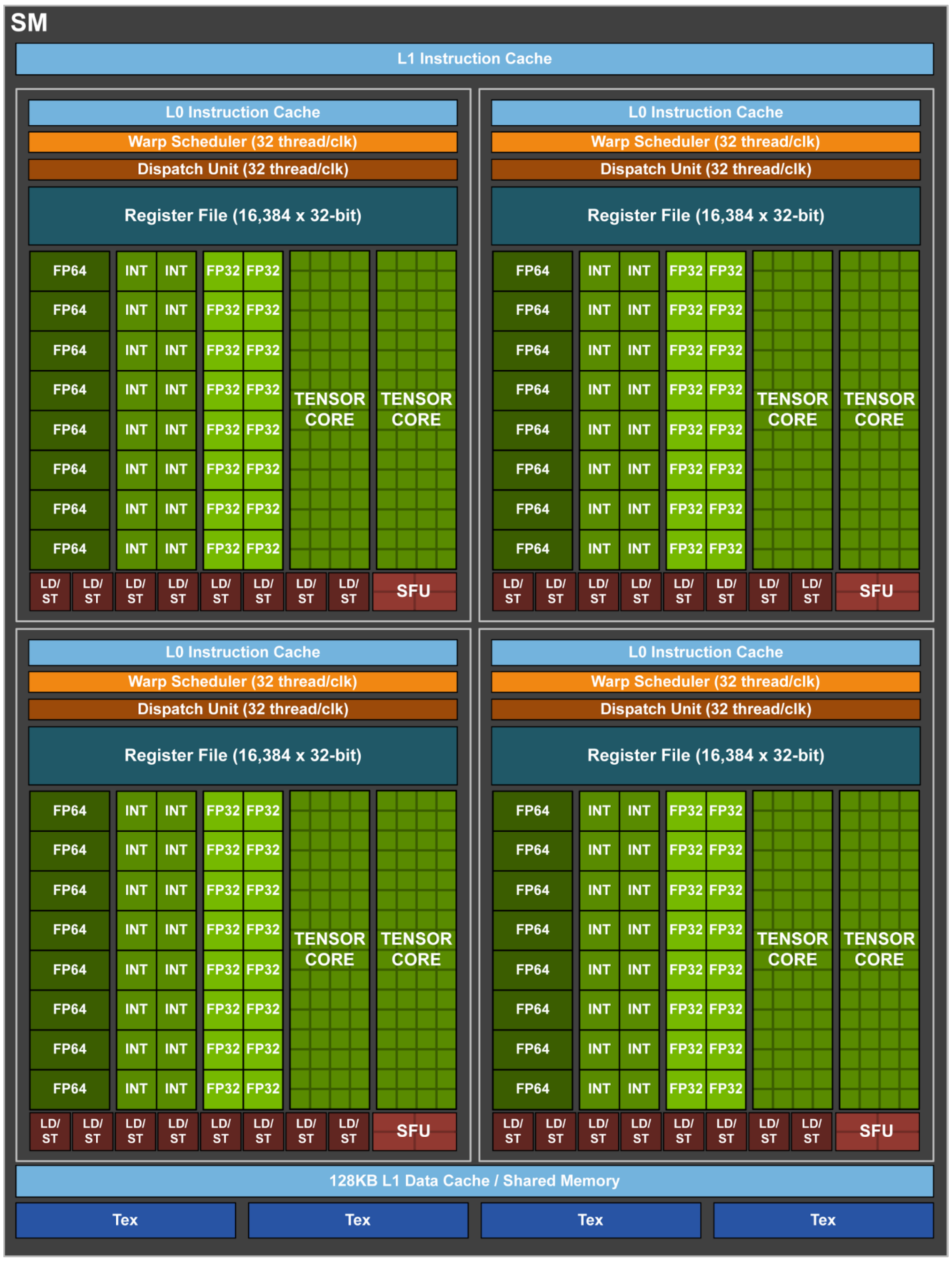

SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
FP64
INT
FP32
TENSOR CORE
LD/ST
SFU
128KB L1 Data Cache / Shared Memory
Tex

Figure: Volta GPU architecture showing CUDA and Tensor Cores within SM [105]

# Appendix D

# Features description for Muon Chambers data

| | **CSC** | **RPC** | **GEM** |
|---|---|---|---|
| **STATIONS** | ME 1/1-ME1/2-ME 2/X -ME 3/X-ME 4/X | RE1-RE2 RE3-RE4 | GE 1/1- GE 2/1-ME 0 |
| **# OF STATIONS** | 5 | 4 | 3 |

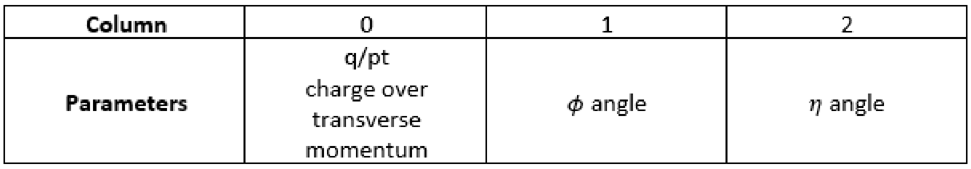
Table: Station distributions in the muon chamber

| **Features** | Phi angle | Theta angle | Bend angle | Time | Ring | Front/ rear | Mask |
|---|---|---|---|---|---|---|---|

Table: Features to be monitored for each hit

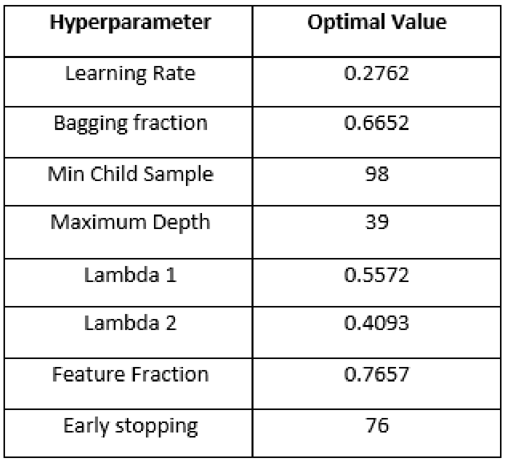
| **Column range** | 0-11 | 12-23 | 24-35 | 36-47 | 48-59 | 60-71 | 72-83 | 84 | 85 | 86 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Particle hit features** | $\phi$ Coordinate | $\theta$ coordinate | Bending angle | Time info | Ring number | Front/ rear hit | Mask | X_road | | |

Table: Dataset format covering the content of the entire 87 columns in the Numpy file 'variables'



| Column | 0 | 1 | 2 |
|---|---|---|---|
| **Parameters** | q/pt charge over transverse momentum | $\phi$ angle | $\eta$ angle |

Table: Dataset format covering the content of the 3 columns in the Numpy file 'parameters'



| Hyperparameter | Optimal Value |
|---|---|
| Learning Rate | 0.2762 |
| Bagging fraction | 0.6652 |
| Min Child Sample | 98 |
| Maximum Depth | 39 |
| Lambda 1 | 0.5572 |
| Lambda 2 | 0.4093 |
| Feature Fraction | 0.7657 |
| Early stopping | 76 |

Table: Bayesian optimization of hyperparameters for the LightGBM model

# Bibliography

[1] J. Allison *et al.*, "Geant4 developments and applications," *IEEE Transactions on Nuclear Science*, vol. 53, no. 1, pp. 270–278, 2006.

[2] J. de Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lemaître, A. Mertens, and M. Selvaggi, "Delphes 3: a modular framework for fast simulation of a generic collider experiment," *Journal of High Energy Physics*, vol. 2014, Feb 2014.

[3] R. Di Sipio, M. F. Giannelli, S. K. Haghighat, and S. Palazzo, "Dijetgan: a generative-adversarial network approach for the simulation of qcd dijet events at the lhc.," *Journal of High Energy Physics, 2019(8), 110.*, 2019.

[4] E. Buhmann, S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, A. Korol, and K. Krüger, "Getting high: High fidelity simulation of high granularity calorimeters with high speed," *arXiv preprint arXiv:2005.05334*, 2020.

[5] H. Qu and L. Gouskos, "Jet tagging via particle clouds," *Physical Review D*, vol. 101, Mar 2020.

[6] E. A. Davis and I. Falconer, *JJ Thompson and the Discovery of the Electron.* CRC Press, 2002.

[7] E. Rutherford, "Lxxix. the scattering of $\alpha$ and $\beta$ particles by matter and the structure of the atom," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 21, no. 125, pp. 669–688, 1911.

[8] S. Hetherton, "How to make your own cloud chamber." https://home.cern/news/news/experiments/how-make-your-own-cloud-chamber, 2015.

[9] A. B. Arbuzov, "Quantum field theory and the electroweak standard model," 2018.

[10] C. Rhodes, "Large hadron collider (lhc)," *Science progress*, vol. 96, pp. 95–109, 03 2013.

[11] "The large hadron collider." https://home.cern/science/accelerators/large-hadron-collider.

[12] F. CERN, "Lhc the guide," 2009.

[13] O. S. Brüning, P. Collier, P. Lebrun, S. Myers, R. Ostojic, J. Poole, and P. Proudlock, *LHC Design Report.* CERN Yellow Reports: Monographs, Geneva: CERN, 2004.

[14] G. Apollinari, O. Brüning, T. Nakamoto, and L. Rossi, "High luminosity large hadron collider hl-lhc," *arXiv preprint arXiv:1705.08830*, 2017.

[15] A. M. Sirunyan, A. Tumasyan, W. Adam, F. Ambrogi, E. Asilar, T. Bergauer, J. Brandstetter, M. Dragicevic, J. Erö, A. E. Del Valle, and et al., "Combined measurements of higgs boson couplings in proton–proton collisions at $\sqrt{s} = 13\,\text{TeV}$," *The European Physical Journal C*, vol. 79, May 2019.

[16] A. Sirunyan, A. Tumasyan, W. Adam, F. Ambrogi, E. Asilar, T. Bergauer, J. Brandstetter, M. Dragicevic, J. Erö, A. Escalante Del Valle, and et al., "Observation of higgs boson decay to bottom quarks," *Physical Review Letters*, vol. 121, Sep 2018.

[17] F. Marcastel, "CERN's Accelerator Complex. La chaîne des accélérateurs du CERN," Oct 2013. General Photo.

[18] H. Jarlett, *What happened while the LHC slept over winter?*, 2017.

[19] "Detector." https://cms.cern/detector.

[20] T. Sakuma and T. McCauley, "Detector and event visualization with sketchup at the cms experiment," *Journal of Physics: Conference Series*, vol. 513, p. 022032, Jun 2014.

[21] S. R. Davis, "Interactive Slice of the CMS detector," Aug 2016.

[22] "Silicon Strip Tracker Performance results 2018," Sep 2018.

[23] *Silicon Strips—CMS Experiment.*

[24] C. Biino, "The CMS electromagnetic calorimeter: overview, lessons learned during run 1 and future projections," *Journal of Physics: Conference Series*, vol. 587, p. 012001, feb 2015.

[25] A. Seiden, "Characteristics of the atlas and cms detectors," *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, vol. 370, pp. 892–906, 02 2012.

[26] *Journal of Instrumentation*, vol. 8, p. P09009–P09009, Sep 2013.

[27] G. Baiatian *et al.*, “Design, Performance, and Calibration of CMS Hadron-Barrel Calorimeter Wedges,” Tech. Rep. CMS-NOTE-2006-138. 1, CERN, Geneva, May 2007.

[28] S. Abdullin, V. Abramov, B. Acharya, M. Adams, N. Akchurin, U. Akgun, E. Anderson, G. Antchev, S. Ayan, S. Aydin, *et al.*, “Design, performance, and calibration of cms hadron-barrel calorimeter wedges,” *The European Physical Journal C*, vol. 55, no. 1, pp. 159–171, 2008.

[29] P. Kumari *et al.*, “Improved-RPC for the CMS muon system upgrade for the HL-LHC,” *JINST*, vol. 15, p. C11012. 11 p, May 2020.

[30] V. Khachatryan, A. Sirunyan, A. Tumasyan, W. Adam, E. Asilar, T. Bergauer, J. Brandstetter, E. Brondolin, M. Dragicevic, J. Erö, and et al., “The cms trigger system,” *Journal of Instrumentation*, vol. 12, p. P01020–P01020, Jan 2017.

[31] D. Acosta, A. Brinkerhoff, E. Busch, A. Carnes, I. Furic, S. Gleyzer, K. Kotov, J. F. Low, A. Madorsky, J. Rorie, B. Scurlock, and W. S. and, “Boosted decision trees in the level-1 muon endcap trigger at CMS,” *Journal of Physics: Conference Series*, vol. 1085, p. 042042, sep 2018.

[32] A. Sirunyan, A. Tumasyan, W. Adam, F. Ambrogi, E. Asilar, T. Bergauer, J. Brandstetter, E. Brondolin, M. Dragicevic, J. Erö, and et al., “Identification of heavy-flavour jets with the cms detector in pp collisions at 13 tev,” *Journal of Instrumentation*, vol. 13, p. P05011–P05011, May 2018.

[33] R. Aaij, B. Adeva, M. Adinolfi, Z. Ajaltouni, S. Akar, J. Albrecht, F. Alessio, M. Alexander, A. Alfonso Albero, S. Ali, and et al., “Search for dark photons produced in 13 tev pp collisions,” *Physical Review Letters*, vol. 120, Feb 2018.

[34] “Observation of the rare bs0 →µ+µ decay from the combined analysis of cms and lhcb data,” *Nature*, vol. 522, p. 68–72, May 2015.

[35] A. Radovic, M. Williams, D. Rousseau, M. Kagan, D. Bonacorsi, A. Himmel, A. Aurisano, K. Terao, and T. Wongjirad, “Machine learning at the energy and intensity frontiers of particle physics,” *Nature*, vol. 560, no. 7716, pp. 41–48, 2018.

[36] Z. Xiong, D. Wang, X. Liu, F. Zhong, X. Wan, X. Li, Z. Li, X. Luo, K. Chen, H. Jiang, and M. Zheng, “Pushing the boundaries of molecular representation for drug discovery with graph attention mechanism,” *Journal of Medicinal Chemistry*, vol. 63, 08 2019.

[37] M. Simonovsky and N. Komodakis, "Graphvae: Towards generation of small graphs using variational autoencoders," 2018.

[38] P. Voigtlaender, M. Krause, A. Osep, J. Luiten, B. B. G. Sekar, A. Geiger, and B. Leibe, "Mots: Multi-object tracking and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7942–7951, 2019.

[39] M. Oleksowicz, "Aristotle on the heart and brain," *European Journal of Science and Theology*, vol. 14, pp. 77–94, 06 2018.

[40] M. Costandi, "The discovery of the neuron," *History of Neuroscience, Neuroscience, Tuesday*, 2006.

[41] B. Mehlig, "Artificial neural networks," 2019.

[42] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 01 2012.

[43] J. Nagi, F. Ducatelle, G. Di Caro, D. Ciresan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella, "Max-pooling convolutional neural networks for vision-based hand gesture recognition," pp. 342–347, 11 2011.

[44] A. Amini, A. Soleimany, S. Karaman, and D. Rus, "Spatial uncertainty sampling for end-to-end control," 2019.

[45] J. Zhang, "Gradient descent based optimization algorithms for deep learning models training," 2019.

[46] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[47] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2014.

[48] G. Karamanolakis, K. R. Cherian, A. R. Narayan, J. Yuan, D. Tang, and T. Jebara, "Item recommendation with variational autoencoders and heterogeneous priors," in *Proceedings of the 3rd Workshop on Deep Learning for Recommender Systems*, pp. 10–14, 2018.

[49] G. Louppe, K. Cho, C. Becot, and K. Cranmer, "Qcd-aware recursive neural networks for jet physics," *Journal of High Energy Physics*, vol. 2019, Jan 2019.

[50] J. Barnard, E. N. Dawe, M. J. Dolan, and N. Rajcic, "Parton shower uncertainties in jet substructure analyses with deep neural networks," *Physical Review D*, vol. 95, Jan 2017.

[51] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," 2013.

[52] M. Cacciari and G. P. Salam, "Dispelling the n3 myth for the kt jet-finder," *Physics Letters B*, vol. 641, no. 1, pp. 57–61, 2006.

[53] M. Andrews, J. Alison, S. An, B. Burkle, S. Gleyzer, M. Narain, M. Paulini, B. Poczos, and E. Usai, "End-to-end jet classification of quarks and gluons with the cms open data," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 977, p. 164304, Oct 2020.

[54] P. T. Komiske, E. M. Metodiev, and M. D. Schwartz, "Deep learning in color: towards automated quark/gluon jet discrimination," *Journal of High Energy Physics*, vol. 2017, Jan 2017.

[55] A. Aurisano, A. Radovic, D. Rocco, A. Himmel, M. Messier, E. Niner, G. Pawloski, F. Psihas, A. Sousa, and P. Vahle, "A convolutional neural network neutrino event classifier," *Journal of Instrumentation*, vol. 11, p. P09001–P09001, Sep 2016.

[56] P. T. Komiske, E. M. Metodiev, B. Nachman, and M. D. Schwartz, "Pileup mitigation with machine learning (pumml)," *Journal of High Energy Physics*, vol. 2017, Dec 2017.

[57] S. Farrell, P. Calafiura, M. Mudigonda, Prabhat, D. Anderson, J.-R. Vlimant, S. Zheng, J. Bendavid, M. Spiropulu, G. Cerati, L. Gray, J. Kowalkowski, P. Spentzouris, and A. Tsaris, "Novel deep learning methods for track reconstruction," 2018.

[58] S. Liechti, "Particle track reconstruction using a recurrent neural network at the $\mu$- 3e experiment," 2018.

[59] D. Guest, K. Cranmer, and D. Whiteson, "Deep learning and its application to lhc physics," *Annual Review of Nuclear and Particle Science*, vol. 68, p. 161–181, Oct 2018.

[60] M. Andrews, M. Paulini, S. Gleyzer, and B. Poczos, "End-to-end physics event classification with the cms open data: Applying image-based deep learning on detector data to directly classify collision events at the lhc," *Computing and Software for Big Science*, vol. 4, no. 1, pp. 1–14, 2020.

[61] K. Albertsson, P. Altoe, D. Anderson, J. Anderson, M. Andrews, J. P. A. Espinosa, and D. ... Bonacorsi, "Machine learning in high energy physics community white paper," *arXiv preprint arXiv:1807.02876*, 2018.

[62] P. Musella and F. Pandolfi, "Fast and accurate simulation of particle detectors using generative adversarial networks," *Comput Softw Big Sci 2: 8*, 2018.

[63] T. Danel, P. Spurek, J. Tabor, M. Smieja, L. Struski, A. Slowik, and L. Maziarka, "Spatial graph convolutional networks," *arXiv e-prints, arXiv-1909*, 2019.

[64] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, p. 1–21, 2020.

[65] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, p. 18–42, Jul 2017.

[66] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery  Data Mining*, Jul 2018.

[67] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017.

[68] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," 2017.

[69] S. Rhee, S. Seo, and S. Kim, "Hybrid approach of relation network and localized graph convolutional filtering for breast cancer subtype classification," 2018.

[70] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[71] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," 2019.

[72] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," 2019.

[73] F. Diehl, "Edge contraction pooling for graph neural networks," 2019.

[74] T. N. Kipf and M. Welling, “Variational graph auto-encoders,” 2016.

[75] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, “Collective classification in network data,” *AI Magazine*, vol. 29, p. 93, Sep. 2008.

[76] B. Perozzi, R. Al-Rfou, and S. Skiena, “Deepwalk,” *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD ’14*, 2014.

[77] L. Tang and H. Liu, “Leveraging social media networks for classification,” *Data Min. Knowl. Discov.*, vol. 23, pp. 447–478, 11 2011.

[78] L. Ruddigkeit, R. Deursen, L. Blum, and J.-L. Reymond, “Enumeration of 166 billion organic small molecules in the chemical universe database gdb-17,” *Journal of chemical information and modeling*, vol. 52, 10 2012.

[79] T. Sterling and J. Irwin, “Zinc 15 - ligand discovery for everyone,” *Journal of chemical information and modeling*, vol. 55, 10 2015.

[80] “Graph auto-encoders.” https://github.com/tkipf/gae.

[81] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “Pointnet: Deep learning on point sets for 3d classification and segmentation,” 2017.

[82] L. Yi, L. Guibas, V. Kim, D. Ceylan, I.-C. Shen, M. Yan, H. Su, A. Lu, Q. Huang, and A. Sheffer, “A scalable active framework for region annotation in 3d shape collections,” *ACM Transactions on Graphics*, vol. 35, pp. 1–12, 11 2016.

[83] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “Pointnet++: Deep hierarchical feature learning on point sets in a metric space,” 2017.

[84] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, “Dynamic graph cnn for learning on point clouds,” 2019.

[85] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, “3d shapenets: A deep representation for volumetric shapes,” 2015.

[86] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, “Pointcnn: Convolution on $\mathcal{X}$-transformed points,” 2018.

[87] G. Bouritsas, S. Bokhnyak, S. Ploumpis, M. Bronstein, and S. Zafeiriou, “Neural 3d morphable models: Spiral convolutional networks for 3d shape representation learning and generation,” 2019.

[88] G. Kasieczka, T. Plehn, J. Thompson, and M. Russel, “Top quark tagging reference dataset,” Mar. 2019.

[89] P. T. Komiske, E. M. Metodiev, and J. Thaler, "Energy flow networks: deep sets for particle jets," *Journal of High Energy Physics*, vol. 2019, Jan 2019.

[90] J. A. Martinez, O. Cerri, M. Pierini, M. Spiropulu, and J.-R. Vlimant, "Pileup mitigation at the large hadron collider with graph neural networks," 2019.

[91] X. Ju, S. Farrell, P. Calafiura, D. Murnane, Prabhat, L. Gray, T. Klijnsma, K. Pedro, G. Cerati, J. Kowalkowski, G. Perdue, P. Spentzouris, N. Tran, J.-R. Vlimant, A. Zlokapa, J. Pata, M. Spiropulu, S. An, A. Aurisano, J. Hewes, A. Tsaris, K. Terao, and T. Usher, "Graph neural networks for particle reconstruction in high energy physics detectors," 2020.

[92] S. R. Qasim, J. Kieseler, Y. Iiyama, and M. Pierini, "Learning representations of irregular particle-detector geometry with distance-weighted graph networks," *The European Physical Journal C*, vol. 79, Jul 2019.

[93] D. Bertolini, P. Harris, M. Low, and N. Tran, "Pileup per particle identification," *Journal of High Energy Physics*, vol. 2014, Oct 2014.

[94] P. W. Battaglia, R. Pascanu, M. Lai, D. Rezende, and K. Kavukcuoglu, "Interaction networks for learning about objects, relations and physics," 2016.

[95] G. Brooijmans *et al.*, "Les Houches 2015: Physics at TeV colliders - new physics working group report," in *9th Les Houches Workshop on Physics at TeV Colliders*, 5 2016.

[96] S. Agostinelli *et al.*, "Geant4—a simulation toolkit," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 506, no. 3, pp. 250 – 303, 2003.

[97] B. Knuteson, "Systematic analysis of hep collider data," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 534, p. 7–14, Nov 2004.

[98] C. O. D. Portal, "http://opendata.cern.ch."

[99] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," 2018.

[100] F. M. Bianchi, D. Grattarola, and C. Alippi, "Spectral clustering with graph neural networks for graph pooling," 2020.

[101] X. Y. Stella and J. Shi, "Multiclass spectral clustering," in *null*, p. 313, IEEE, 2003.

[102] P. T. Komiske, R. Mastandrea, E. M. Metodiev, P. Naik, and J. Thaler, "Exploring the space of jets with cms open data," *Physical Review D*, vol. 101, Feb 2020.

[103] "Helmholtz GPU Hackathon 2020, howpublished = https://www.fz-juelich.de/shareddocs/termine/ias/jsc/en/events/2020/helmholtz-gpu-hackathon-2020.html: :text=the

[104] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in tensorflow," *arXiv preprint arXiv:1802.05799*, 2018.

[105] M. H. N. S. Luke Durant, Olivier Giroux, *Inside Volta: The World's Most Advanced Data Center GPU*, 2017.

[106] *NVIDIA TESLA V100 GPU ACCELERATOR.*

[107] *The CMS electromagnetic calorimeter project: Technical Design Report.* Technical Design Report CMS, Geneva: CERN, 1997.