ALMA MATER STUDIORUM · UNIVERSITY OF BOLOGNA

School of Science
Department of Physics and Astronomy
Master Degree in Physics

A Variational Autoencoder Application for real-time Anomaly Detection at CMS

Supervisor:

Prof. Daniele Bonacorsi

Co-supervisor:

Dr. Luca Anzalone
Dr. Marco Lorusso

Submitted by:
Lorenzo Valente

Academic Year 2021/2022

Contents

Abstract

Despite providing invaluable data in the field of High Energy Physics, towards higher luminosity runs the Large Hadron Collider (LHC) will face challenges in discovering interesting results through conventional methods used in previous run periods. Among the proposed approaches, the one we focus on in this thesis work – in collaboration with CERN teams, involves the use of a joint variational autoencoder (JointVAE) machine learning model, trained on known physics processes to identify anomalous events that correspond to previously unidentified physics signatures. By doing so, this method does not rely on any specific new physics signatures and can detect anomalous events in an unsupervised manner, complementing the traditional LHC search tactics that rely on model-dependent hypothesis testing. The algorithm produces a list of anomalous events, which experimental collaborations will examine and eventually confirm as new physics phenomena. Furthermore, repetitive event topologies in the dataset can inspire new physics model building and experimental searches. Implementing this algorithm in the trigger system of LHC experiments can detect previously unnoticed anomalous events, thus broadening the discovery potential of the LHC.
This thesis presents a method for implementing the JointVAE model, for real-time anomaly detection in the Compact Muon Solenoid (CMS) experiment. Among the challenges of implementing machine learning models in fast applications, such as the trigger system of the LHC experiments, low latency and reduced resource consumption are essential. Therefore, the JointVAE model has been studied for its implementation feasibility in Field-Programmable Gate Arrays (FPGAs), utilizing a tool based on High-Level Synthesis (HLS) named HLS4ML. The tool, combined with the quantization of neural networks, will reduce the model size, latency, and energy consumption.

Chapter 1 provides an overview of the experimental apparatus, starting with a detailed description of the LHC accelerator machine. This is followed by a presentation of the key features of the CMS detector, which is essential for the analysis.

Chapter 2 is dedicated to introducing Machine and Deep Learning concepts relevant to the anomaly detection task explored in the following chapters. The chapter focuses on the architecture of neural networks, specifically those models used in anomaly detection, such as the variational autoencoder, and covers related concepts, operations and terminology. The chapter also offers an overview of the implementation of these networks using popular frameworks widely used in scientific domains, as well as various compression techniques for hardware optimization of deep learning models for fast inference.

Chapter 3 presents the original outcomes of this project, which includes the development and design of a JointVAE model optimized for detecting anomalies in image datasets of

QCD and top jets. The chapter provides a detailed description of the exploration in the hyperparameter space to build the optimal JointVAE, and the reduction of the model size by minimizing the parameters in the trade-off between size and final performance, and final quantization for the FPGA target. In addition, a study is conducted on the feasibility of implementing the quantized model on an FPGA target, as well as an evaluation of the estimated resource consumption.

Chapter 1

The CMS experiment at the LHC

The purpose of this chapter is to give a comprehensive understanding of the major characteristics of the Large Hadron Collider (LHC) and the Compact Muon Solenoid (CMS) experiment, including its subdetectors. This information is crucial as the research analysis presented in the final chapter of this thesis is conducted within the context of the CMS experiment.

1.1 The Large Hadron Collider

The Large Hadron Collider (LHC) [1, 2] is the world's largest and most powerful particle accelerator. It is located at CERN, the European Organization for Nuclear Research, near Geneva, Switzerland. The LHC is a collaborative project involving thousands of scientists from around the world.

The LHC is a circular tunnel that spans over 27 kilometres (17 miles) in circumference and is buried deep underground. It is designed to accelerate particles up to 14 TeV, to nearly the speed of light and then collide them together to study the fundamental structure of matter and the forces that shape the universe.

The LHC operates by accelerating two beams of particles in opposite directions around the circular tunnel and then bringing them into collision at four points around the circumference where the two rings of the machine intersect; these are in correspondence with the four particle detectors: ALICE, ATLAS, CMS and LHCb, as it is shown in figure 1.1. These collisions produce new particles that are detected and analyzed by the experiments at the LHC. The data collected by these experiments are used to study a wide range of phenomena, including the Higgs boson, the nature of dark matter, and the fundamental nature of the universe.

In the LHC ring, beam injection and gradual acceleration of protons take place in many stages:

1. the process of stripping hydrogen atoms of their orbital electrons produces protons;

2. protons begin to accelerate, reaching energies of up to 50 MeV thanks to the LINAC2 linear accelerator;
3. protons are added to the Proton Synchrotron Booster (PSB), where the beam's energy reaches around 1.4 GeV;
4. protons are then accelerated to a maximum of 25 GeV in the Proton Synchrotron (PS);
5. the Super Proton Synchrotron (SPS), where they reach an energy of 450 GeV, receives the proton beam;
6. finally, protons are transferred into the two neighbouring and parallel beam pipes of the LHC in a bunch configuration. They then circulate for several hours around the ring, one beam moving in a clockwise direction and the other in an anticlockwise one.

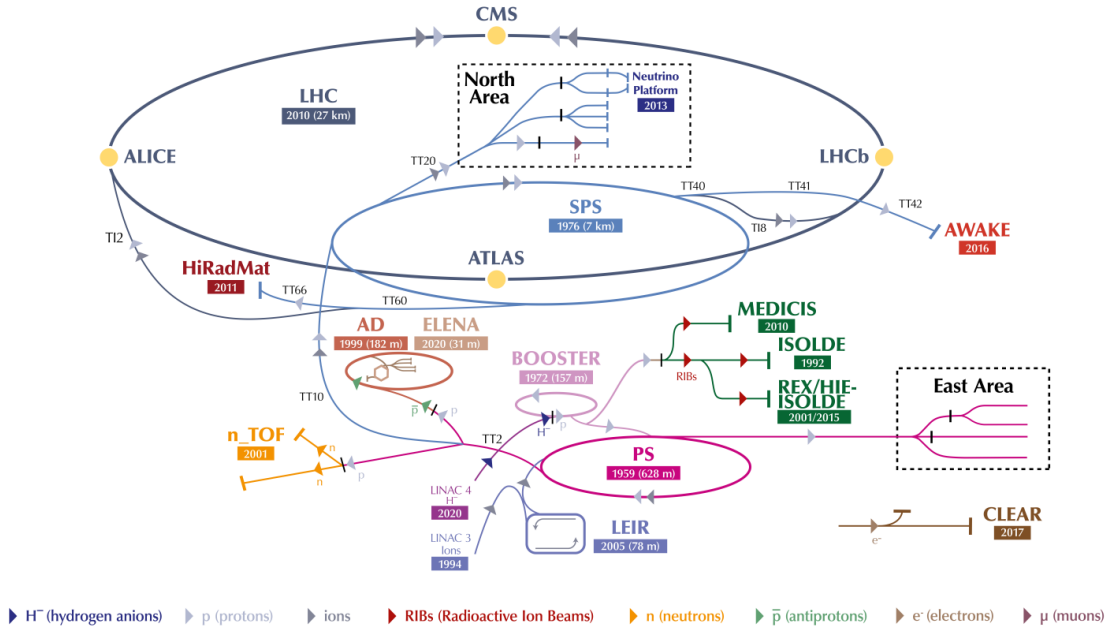


Figure 1.1: CERN accelerator complex in schematic form [3].

Table 1.1 lists some of the main technical parameters of the Large Hadron Collider (LHC).

1.1.1 The vacuum system

The vacuum system [4] of the LHC is a critical component of the accelerator that is used to evacuate the air and other contaminants from the accelerator ring. This is necessary because the protons or ions in the beam must travel through a vacuum to achieve the high energies required for the experiments. The vacuum system also helps to minimize the amount of energy lost by the beam as it travels through the accelerator. The vacuum is equivalent to pressure to the order of $10^{-13} atm$.

The LHC vacuum system consists of a series of pipes and chambers that are installed along the length of the accelerator ring. These pipes and chambers are connected to pumps

Quantity	Value
Circumference (m)	26659
Magnets working temperature (K)	1.9
Number of magnets	9593
Number of principal dipoles	1312
Number of principal quadrupoles	392
Number of radio-frequency cavities per beam	16
Nominal energy, protons (TeV)	6.5
Nominal energy, ions (TeV/nucleon)	2.76
Magnetic field maximum intensity (T)	8.33
Project luminosity ($cm^{-2}s^{-1}$)	2.06×10^{34}
Number of proton packages per beam	2808
Number of proton per package (outgoing)	1.1×10^{11}
Minimum distance between packages (m)	$\sim$7
Number of rotations per second	11 245
Number of collisions per crossing (nominal)	$\sim$20
Number of collisions per second (millions)	600

Table 1.1: The main technical parameters of the Large Hadron Collider.

and other equipment that are used to create and maintain the vacuum. The vacuum system is divided into several different sections, each of which serves a specific purpose.

One of the main components of the LHC vacuum system is the beam pipe, which is a tube that runs the entire length of the accelerator ring and encloses the beam of particles. The beam pipe is made of special materials that can withstand the high temperatures and radiation levels generated by the beam. The inside of the beam pipe is coated with a thin layer of metal, which helps to reflect the beam and prevent it from dissipating. The vacuum system also includes a series of pumping stations that are used to evacuate the air from the beam pipe and other parts of the accelerator. These pumping stations are equipped with a variety of pumps, including cryogenic, turbomolecular, and ion pumps. These pumps operate at different pressures and temperatures and are used to create and maintain the vacuum.

The LHC vacuum system is monitored and controlled by a sophisticated computer system that is used to ensure that the vacuum is maintained at the correct level. The system is also used to detect any problems or issues with the vacuum and to take corrective action as needed.

1.1.2 Electromagnets

Electromagnets [5] guide and focus the beams of particles as they travel through the accelerator. These electromagnets are superconducting, meaning that they can conduct electricity with zero resistance when cooled to very low temperatures. There are two types of electromagnets used in the LHC: dipole magnets and quadrupole magnets. Dipole magnets (Figure 1.2) are used to bend the beam of particles, while quadrupole magnets are used to focus the beam.

Both types of magnets use superconducting coils made of niobium-titanium wire to

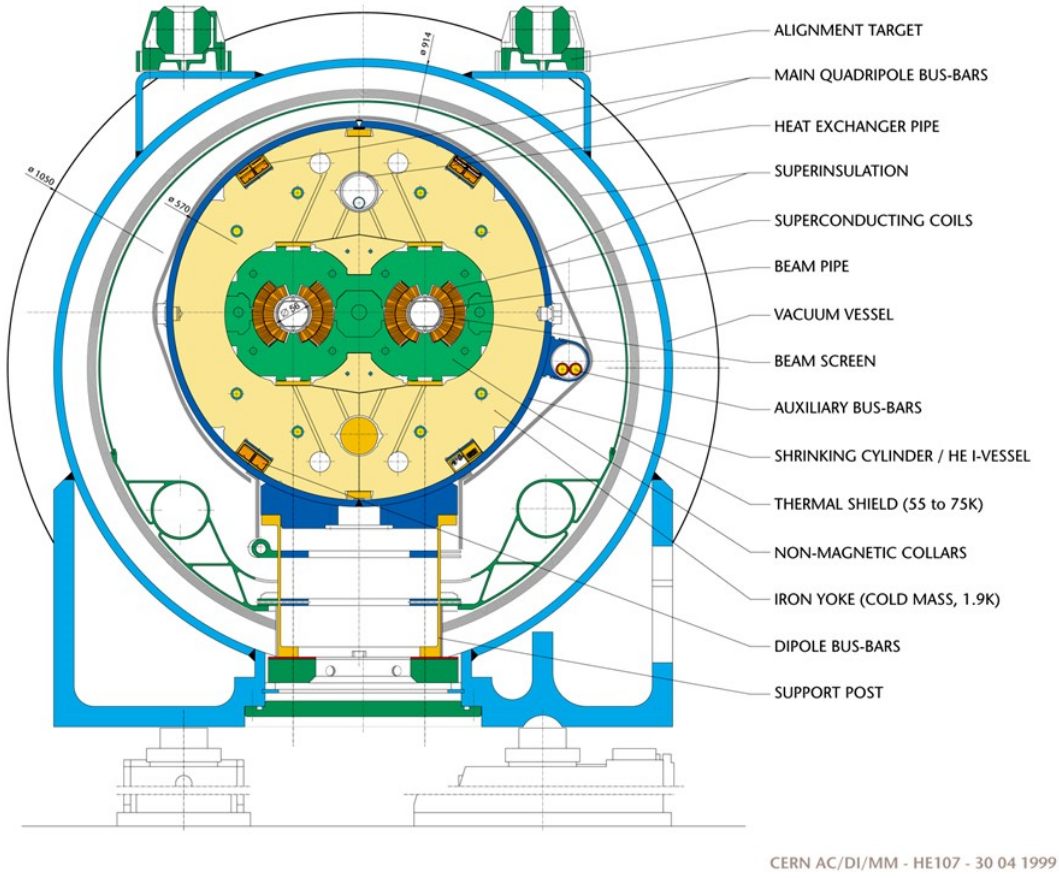


Figure 1.2: Cross section of LHC dipole [6].

generate the magnetic field. These magnets are arranged in a lattice configuration along the accelerator ring and are used to control the path of the beam. The magnets are powered by electrical current and are cooled to a temperature of about 1.9 K (-271.3°C) using liquid helium. The magnets in the LHC are some of the most powerful electromagnets in the world, generating a magnetic field of about 8.3 tesla, which is about 100,000 times stronger than the Earth's magnetic field. The magnetic field is used to steer the beam of particles and keep it on course as it travels through the accelerator. The intensity of the magnetic field required to bend protons in the LHC accelerator can be calculated using the following equation:

$$p[TeV] = 0.3 \times B[T] \times r[km], \tag{1.1}$$

where p is the momentum of the beam particle and r is the radius of the LHC ring.

1.1.3 Radiofrequency Cavities and Luminosity

Radiofrequency (RF) cavities [1, 7] are an essential component of the Large Hadron Collider accelerator. They are used to accelerate the beams of protons or heavy ions as they travel through the accelerator. The RF cavities are located along the length of the accelerator and are powered by high-voltage generators. Each RF cavity consists of a hollow metal tube that is resonant at a specific frequency. When the RF generator is turned on, it generates an electromagnetic field inside the cavity. This field oscillates at

the same frequency as the cavity, causing the particles in the beam to oscillate as well. As the particles oscillate, they gain energy from the electromagnetic field. This process is known as "acceleration by resonant coupling."

In the LHC, 16 RF cavities are placed inside four cylindrical refrigerators called *cryomodules*. These cryomodules enable the RF cavities to operate in a superconducting state, which is necessary for their correct functioning. In RF cavities, power is applied sequentially, with the first cavity providing initial acceleration, followed by subsequent cavities providing additional acceleration. The RF cavities operate at a frequency of 400 MHz and are capable of accelerating the particles to very high energies. The cavities work at a temperature of 4.5K and are organized into four cryomodules. When the accelerator is operating under normal conditions, each proton beam is divided into 2808 bunches, each containing approximately 10^{11} protons. These bunches are a few centimetres long and 1 mm wide when they are far from the collision point, but they are compressed down to a size of 16 nm near the collision point. This increases the probability of a proton-proton collision. The RF cavities in the LHC are a crucial part of the accelerator and play a key role in the success of the experiments conducted at the facility. They are carefully designed and manufactured to meet the demanding requirements of the accelerator, and they are monitored and controlled by a sophisticated computer system.

One of the main factors that determine the *instantaneous luminosity*[1] The instantaneous luminosity is typically measured in inverse femtobarns per second ($fb^{-1}s^{-1}$). It is calculated based on the number of protons in each beam, the number of bunches in each beam, the cross-section of the particles, and the bunch spacing. The LHC uses beams of protons or heavy ions that have intensities of about 1.1×10^{11} or 3×10^{9} particles per beam, respectively. The beam intensity is limited by the strength of the superconducting magnets that are used to guide and focus the beams, as well as the RF cavities that are used to accelerate the particles. Another factor that affects the instantaneous luminosity of the LHC is the bunch spacing, which is the distance between the bunches of particles in the beam. The LHC uses a bunch spacing of about 25 ns, which allows the bunches to pass through the accelerator without colliding with each other. The total amount of data collected by the Large Hadron Collider over some time is the *integrated luminosity*. It is measured in inverse femtobarns (fb^{-1}) and is an important parameter because it determines the statistical significance of the results. Run 1 (2009-2013) and Run 2 (2015-2018) delivered 200 fb^{-1} [8] for the two general purpose experiments, ATLAS and CMS (only 5% of the total integrated luminosity to be collected). The integrated luminosity of the LHC is constantly increasing as more data is collected.

1.2 LHC detectors

The LHC accelerates beams of protons or heavy ions to very high energies and collides them at various points along the accelerator ring. The collisions produce a variety of subatomic particles, which are detected by a series of detectors located around the accelerator. Each detector is designed to measure different aspects of particle collisions and is optimized for studying specific types of particles or phenomena. There are four main

[1] *instantaneous luminosity* is a measure of the number of particle collisions that occur per unit of time in a particle accelerator. It is an important parameter for experiments conducted at the accelerator because it determines the rate at which data can be collected.

detectors at the LHC:

- **ALICE** (A Large Ion Collider Experiment);
- **ATLAS** (A Toroidal LHC ApparatuS);
- **CMS** (Compact Muon Solenoid);
- **LHCb** (Large Hadron Collider beauty).

In the following sections, the LHC detectors will be briefly introduced, with a particular focus on the CMS experiment.

1.2.1 ALICE

The ALICE [9] is a detector located at the location of St. Genis-Pouilly, France. It is designed to study the properties of the quark-gluon plasma, a state of matter that is thought to have existed shortly after the Big Bang and to study the properties of heavy ions. ALICE is composed of several subdetectors that are used to measure various aspects of particle collisions. These include a central barrel detector, which is used to detect charged particles, and a forward detector, which is used to detect neutrons and photons.

One of the main goals of the ALICE experiment is to study the properties of the quark-gluon plasma, which is a state of matter that is thought to have existed in the early universe. The quark-gluon plasma (QGP) is created by colliding heavy ions at very high energies, and it is believed to be composed of quarks and gluons that are not bound together in the same way as they are in normal matter.

In addition, to studying the QGP, the ALICE experiment is also used to study the properties of heavy ions, which are atoms with a large number of protons and neutrons in their nucleus. By colliding these ions at very high energies, scientists can study the properties of the nucleus and learn more about the fundamental forces that govern the behaviour of the particles inside it.

1.2.2 ATLAS

The ATLAS experiment [10] is a detector located at the location of Meyrin, Switzerland. It is a general-purpose detector that is used to study a wide range of phenomena, including the Higgs boson, dark matter, and new physics beyond the Standard Model (SM). The ATLAS experiment is composed of several subdetectors that are used to measure various aspects of particle collisions. These include a central tracker, which is used to measure the momentum of charged particles, and a calorimeter, which is used to measure the energy of particles.

One of the main goals of the ATLAS experiment is to study the Higgs boson, which is a particle that was predicted by the Standard Model of particle physics and was discovered at the LHC in 2012. The Higgs boson is believed to be responsible for giving particles mass, and by studying it, scientists hope to learn more about the fundamental forces that govern the behaviour of the universe. In addition to studying the Higgs boson, the ATLAS experiment is also used to search for dark matter, which is a mysterious form of matter that is thought to make up about 85% of the mass in the universe. By studying the collisions of

particles at the LHC, scientists hope to find evidence of dark matter and learn more about its properties.

Finally, the ATLAS experiment is also used to search for new physics beyond the Standard Model, which is the current theoretical framework that explains the fundamental forces and particles in the universe. By studying the collisions at the LHC, scientists hope to find evidence of new particles or phenomena that could help to improve our understanding of the universe and the laws of nature.

1.2.3 CMS

The CMS experiment [11] is a general-purpose detector located at the location of Cessy, France. It is built around a large solenoid magnet with a cylindrical shape that is capable of generating a strong magnetic field. Like the ATLAS experiment, CMS is used to study a wide range of phenomena, including the Higgs boson, dark matter, and new physics beyond the Standard Model.

1.2.4 LHCb

The LHCb experiment [12] is a detector located at the location of Ferney-Voltaire, France. It is optimized for studying the properties of heavy quarks, which are a type of subatomic particle that is heavier than the up and down quarks that make up protons and neutrons, and the asymmetry between matter and antimatter. The LHCb experiment is composed of several subdetectors that are used to measure various aspects of particle collisions. These include a vertex locator, which is used to measure the tracks of charged particles, and a calorimeter, which is used to measure the energy of particles.

1.2.5 Other experiments

There are several other experiments at the LHC in addition to ALICE, ATLAS, CMS, and LHCb. Some examples are described in the following subsections. These are just a few examples of the many experiments that are conducted at the LHC.

TOTEM

The TOTEM (Total Elastic and Inelastic Scattering Measurement) experiment [13] is designed to measure the elastic and inelastic scattering of protons at high energies and is used to study the structure of the proton and the strong force that governs it. It is composed of several subdetectors that are used to measure various aspects of particle collisions. These include Roman Pot detectors, which are used to measure the scattering of protons, and a telescope, which is used to measure the tracks of charged particles.

One of the main goals of the TOTEM experiment is to study the structure of the proton, which is a subatomic particle that is found in the nucleus of atoms. The proton is made up of smaller particles called quarks and gluons, and the TOTEM experiment is used to study the way these particles are arranged within the proton. In addition to studying the proton, the TOTEM experiment is also used to study the strong force, which is the force that holds quarks and gluons together within the proton.

LHCf

The LHCf (Large Hadron Collider forward) experiment [14] is designed to measure the production of neutral particles in proton-proton collisions and is used to study cosmic rays, which are high-energy particles that are thought to be produced by distant objects in the universe, such as supernovae and active galactic nuclei. The LHCf experiment is composed of two detectors that are located at opposite ends of the LHC ring, near the interaction points of the proton beams. These detectors are used to measure the production of neutral particles, such as photons and neutrons, that are produced in collisions.

1.3 The Compact Muon Solenoid experiment

The CMS (Compact Muon Solenoid) experiment is one of the two large multipurpose detector experiments at LHC. The CMS experiment is designed to explore the physics of proton-proton collisions at the TeV scale, with a particular focus on searching for the Higgs boson. It is also designed to study the properties of heavy quarks, such as the top quark and beauty quark, and τ physics at low luminosities, as well as to study the physics of heavy ions, such as in lead ions collisions. It was designed to operate in proton-proton (Pb-Pb) collisions at a center-of-mass energy of 14 TeV (5.5 TeV) and with luminosities up to $10^{34} cm^{-2} s^{-1}$ It is a collaborative effort involving thousands of scientists from around the world. The detector, located underground, consists of several different subdetectors, each designed to measure different aspects of the particles produced in the collisions. Figure 1.3 shows a picture of the cross-section of the CMS detector.

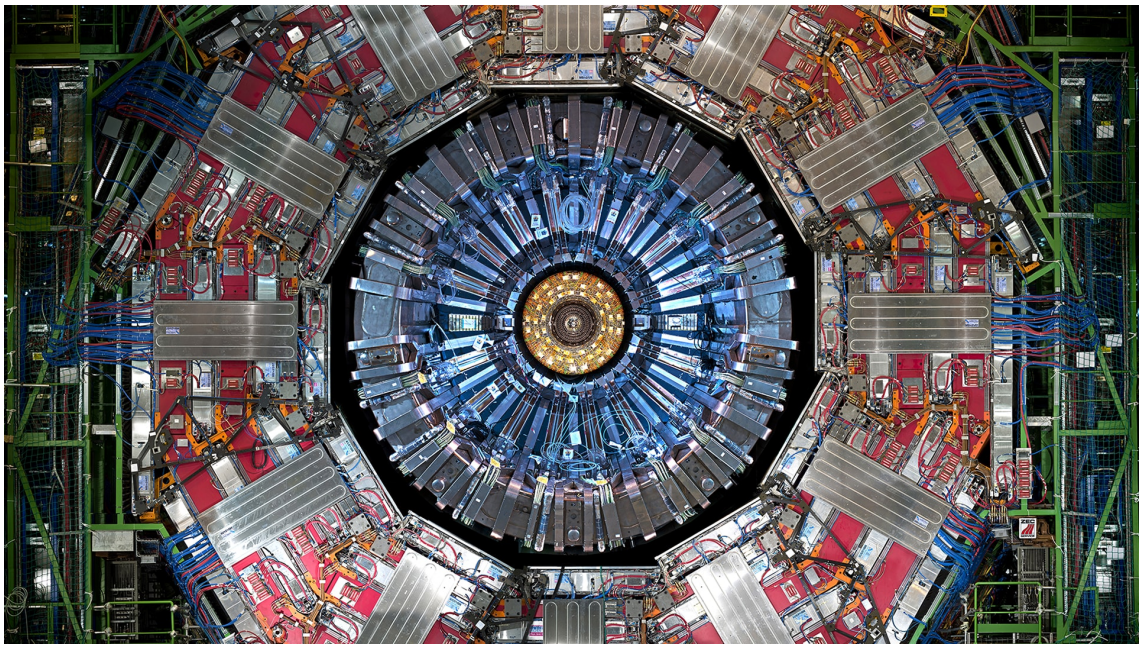


Figure 1.3: A view of CMS experiment.

1.3.1 The detector structure

The CMS detector gets its name from the fact that:

- given the amount of detector material it includes, it is fairly **compact** at 15 meters high and 21 meters long;

- it is optimised to detect **muons** precisely;
- it is equipped with the strongest **solenoid** magnet ever created.

The detector [15] consists of a cylindrical barrel made up of five slices, and two disk-like endcaps. It measures 21.6 meters in length, 15 meters in diameter, and weighs around 12500 tons. The structure is made up of various layers that are designed to track and measure the properties and paths of different types of subatomic particles. The detector is also surrounded by a large solenoid that is based on superconductive technology, operates at a temperature of 4.4K, and generates a magnetic field of 3.8T.

In CMS [16], a right-handed coordinate system is constructed, with the x-axis pointing radially inward to the center of the accelerator ring, the y-axis pointing upward, and the z-axis parallel to the beam pipe as illustrated in Figure 1.4 . This coordinate system is centred at the nominal collision site.

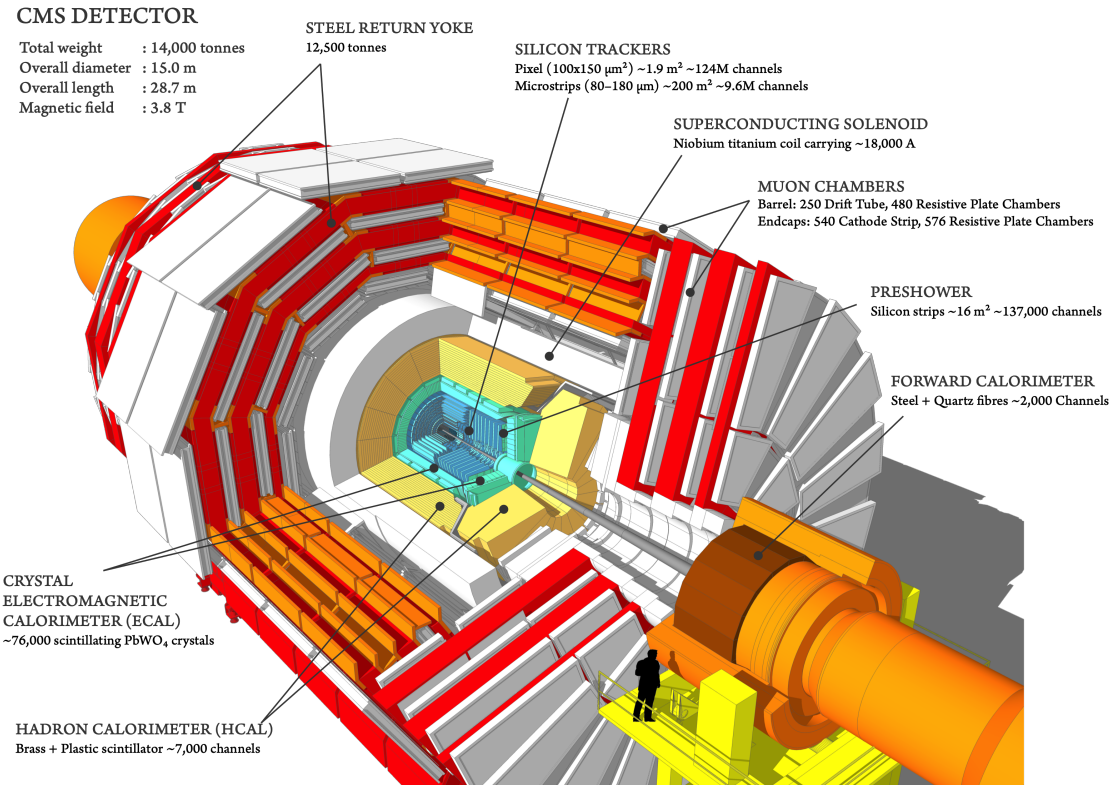


Figure 1.4: The CMS detector's cutaway diagram.

Because of the cylindrical structure of the CMS detector, it is often convenient to use a polar system to describe the four-momentum of particles, as shown in Figure 1.5.

Given the CMS detector's cylindrical design, it is frequently more convenient to characterize the four particle momentums using a polar system: measured from the *z-axis*, the *polar angle* θ in a range $0 \leq \theta \leq \pi$, while the *aximuthal angle* ϕ is measured in teh *x-y plane* from the *x-axis* in a range $0 \leq \phi \leq 2\pi$. In a collision, the center-of-mass experiences an increase in momentum along the *z-axis* as observed from the laboratory frame. Consequently, the coordinates which are typically used to explain the kinematics are (p_T, y, ϕ, m), where m is the *invariant mass*, p_T the *transverse momentum* defined as:

$$p_T = p \sin\theta = \sqrt{p_x^2 + p_y^2}, \tag{1.2}$$

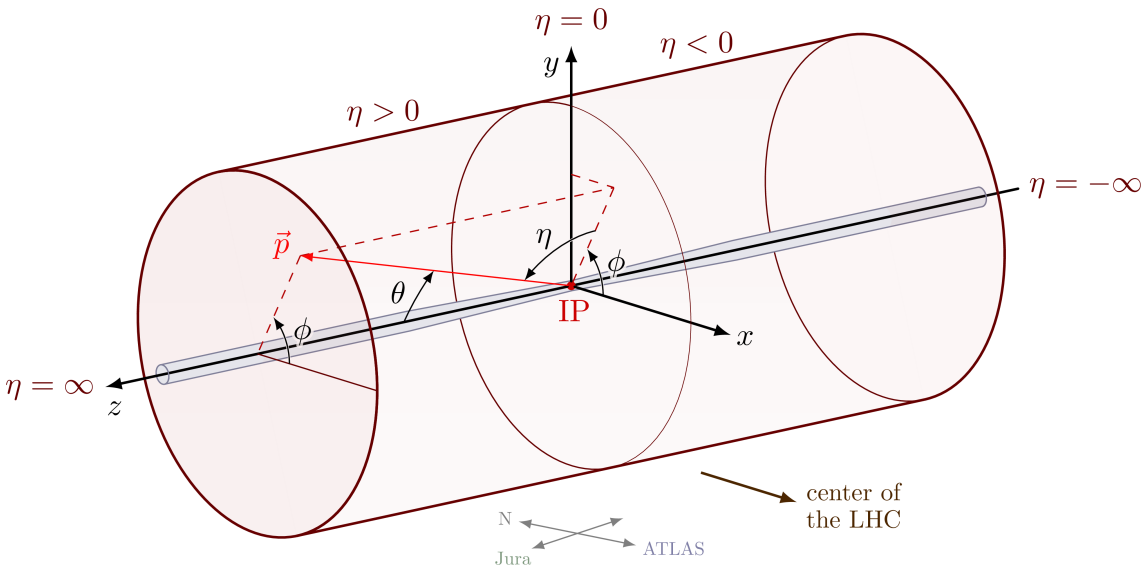


Figure 1.5: CMS coordinate system with the cylindrical detector.

and the *rapidity* defined as:

$$y = \frac{1}{2} \ln \left(\frac{E + p_z}{E - p_z} \right). \tag{1.3}$$

It is possible to define an additional quantity, *pseudorapidity* is a measure of the angle of a particle's motion relative to the beam line. It is defined using the polar angle θ of the particle's momentum vector with the beamline and is given by the equation:

$$\eta = \frac{1}{2} \ln \left(\frac{|p| + p_z}{|p| - p_z} \right). \tag{1.4}$$

Pseudorapidity is useful because it is approximately equal to the particle's rapidity, at ultrarelativistic energies. It is also often used because it increases monotonically with the polar angle for small angles, allowing for easy comparison of the angles of different particles.

CMS detector consists of several layers that are used to identify and measure the properties of particles produced in high-energy collisions. There are three main layers of the CMS detector:

- the innermost layer is the **tracker**, which is used to measure the trajectories of charged particles. It is made up of layers of silicon detectors that can accurately determine the position and momentum of charged particles as they pass through.
- The second layer is the **calorimeters**, which is used to measure the energy of particles. It consists of layers of dense material, such as lead or brass, interleaved with layers of detectors. When a particle passes through the calorimeter, it deposits some of its energy in the material, and this energy is then detected by the detectors.
- The outermost layer is the **muon system**, which is used to identify and measure the properties of muons, which are heavy particles similar to electrons. The muon

system consists of layers of detectors placed at large radii from the beamline to detect the muons as they pass through.

1.3.2 Inner tracking system

The tracking system [17] which measures the momentum of particles by their curvature radius through the magnetic field, is a crucial element in the CMS design. The larger the curvature radius, the greater the momentum of the particles. The Tracker is an effective tool for monitoring the paths of muons, electrons, hadrons, and the decay products of short-lived particles such as beauty quarks. It has a low degree of interference with these particles and is highly resistant to radiation.

At the luminosity level of the LHC, approximately 1000 charged particles are produced from proton-proton interactions every 25 nanoseconds. This high particle production rate necessitates the use of a system with high granularity and radiation resistance to accurately track and measure these particles. To perform efficient and precise measurements, the inner tracking system consists of several layers placed around the collision region, including *silicon pixel detectors* in its innermost part, *silicon strip modules* of the different pitch in its central and external part.

However, this high granularity leads to elevated power consumption and requires efficient cooling infrastructure to function well at low temperatures (around -10°C) and prevent radiation damage. To minimize multiple scattering and other interactions, it is necessary to keep the total amount of material in the tracker as low as possible, so a compromise in the design had to be made. The detector shown in Figure 1.6 is made entirely of silicon and covers a pseudorapidity region of up to $|\eta| < 2.5$ with a radius smaller than 1.2m and $|z| < 2.7$m. This makes it the largest silicon tracker ever built. It has a total of 9.3 million strip sensors and 66 million pixel sensors.

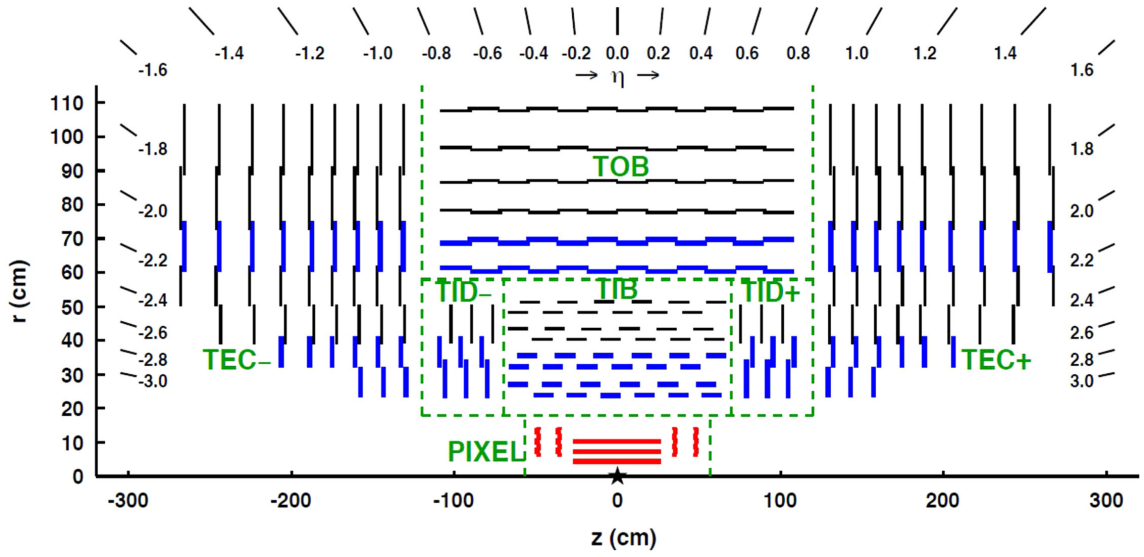


Figure 1.6: A graphical representation of a tracker slice in the r-z plane is shown, with pixel modules in red, single-sided strip modules as black thin lines, and strip stereo modules in blue thick lines [18].

Silicon pixel detectors

Silicon pixel detectors are a key component of the tracking system, these detectors are similar to silicon strip detectors, but with a much finer segmentation of silicon pixels, whose cell size is of $100 \times 150 \mu m^2$, placed on a silicon substrate, allowing for more precise measurement of the particle's trajectory. Like the strip detectors, the pixel detectors are arranged in a matrix of columns and rows. As the particle passes through the detector, it leaves a trail of ionization along its path. This ionization is collected by the detectors and converted into an electrical signal, which is then used to determine a precise 3D vertex reconstruction of the particle. The pixel detectors are sensitive to charged particles with a wide range of energies and can operate in a high-radiation environment. They are an essential component of the CMS tracking system, allowing for the precise measurement of the trajectory and momentum of charged particles produced in high-energy proton-proton and heavy ion collisions.

Silicon strip detectors

The outermost regions of the tracking system are made up of several layers of silicon microstrip detectors. These layers contain approximately 10 million detector strips, which are divided into 15200 modules and monitored by 80000 microelectronic chips on a silicon surface of around 200 square meters. These detectors are capable of detecting the passage of charged particles produced in proton-proton collisions. The four innermost barrel layers of the system make up the Tracker Inner Barrel (TIB) system, while the six outermost layers are called the Tracker Outer Barrel (TOB). The Tracker Inner Disk (TID), consisting of three layers of disks, is located on each side of the TIB. Finally, a group of nine disks perpendicular to the beam axis, placed after the TOB and TID, make up the Tracker End-Cap (TEC). Each module of the tracking system is composed of three parts: sensors, a support structure, and electronics for data acquisition. The sensors have a high response rate and excellent spatial resolution, enabling them to detect many particles in a small space. They detect electrical currents produced by interacting particles and transmit the collected data.

1.3.3 Electromagnetic Calorimeter

The Electromagnetic Calorimeter (ECAL) [19, 20] (see Figure 1.7) is designed to have excellent energy resolution and good position resolution of high-energy photons and electrons produced in the collisions that take place at the CMS experiment collision point, in addition, accurate measurements of hadronic jets will be carried out in combination with the Hadron Calorimeter. Electrons and photons are necessary components in at least three of the Higgs boson decay channels ($H \to \gamma\gamma$, $H \to ZZ \to 4e^{\pm}$, $H \to WW \to e\nu e\nu$), making the performance of the high-resolution CMS ECAL a crucial issue. The design of the ECAL was driven by the key physics channel of $H \to \gamma\gamma$. This decay mode is the most sensitive for a low-mass SM Higgs boson ($m_H < 150GeV$). Despite that, the tracker is only able to identify charged particles with a precision that is inversely proportional to their p_T, the calorimeters can measure both charged and neutral particles with a resolution that is proportional to the increase in the particle's energy. The ECAL design requirements were:

- the tracker coverage is matched by ECAL excellent energy and position/angle resolution up to $|\eta| < 2.5$;

- the system is sealed off from its environment (hermeticity), is small in size or densely packed (compactness), and has a high level of detail or resolution (high granularity);
- the system has a fast response time (about 25 ns) and can measure particle identification, energy, and isolation at the trigger level;
- the system has a large dynamic range (from 5 GeV to 5 TeV) and excellent linearity (at the per-mill level);
- the system has a high level of radiation tolerance, having been designed for 14 TeV and L = 1034 $cm^{-2}s^{-1}$, with a total luminosity of 500/fb.

The 75,848 calorimeter crystals are arranged in a central barrel section (EB), with pseudorapidity coverage up to $|\eta| = 1.48$, closed by two endcaps (EE), extending coverage up to $|\eta| = 3.0$. Crystals are projective and positioned slightly off-pointing (3^0) relative to the interaction point (IP) to avoid cracks aligned with particle trajectories. The calorimeter has no longitudinal segmentation, and the measurement of the photon angle relies on the primary vertex reconstruction from the silicon tracker. A *pre-shower* detector (ES), consisting of lead absorbers and silicon strip sensors (4,288 sensors, 137,216 strips, with an x-y view of 1.90 x 61 mm^2), is placed in front of the endcaps at $1.65 < |\eta| < 2.6$ to improve photon-π^0 separation. The total thickness of the ES is approximately 3 radiation lengths.

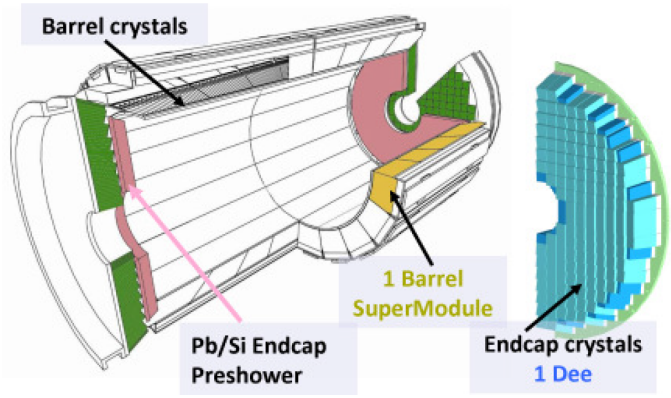


Figure 1.7: A schematic diagram of the CMS electromagnetic calorimeter is shown.

Crystal properties

The CMS ECAL is a hermetic, homogeneous, fine-grained lead tungstate ($PbWO_4$) crystal calorimeter. A homogeneous medium was chosen to minimize sampling fluctuations and improve energy resolution. Dense crystals offer the potential for excellent performance and compactness. The design of the CMS electromagnetic calorimeter allowed it to be placed within the volume of the CMS superconducting solenoid magnet. $PbWO_4$ scintillating crystals have several important features that make them useful for calorimeters. These include a high density ($\delta = 8.28g/cm^3$), a very short radiation length, and a small Molière radius ($X_0 = 0.85$cm and $R_M 2.19$ cm, respectively). These properties allow for the creation of a compact calorimeter with high granularity. The crystals also produce fast signals, with 80% of the light emitted within 25 ns, which is important given the high collision rate of the LHC (40 MHz). The light emission peak is at 420 nm and the crystals are transparent throughout their scintillation emission spectrum. There are a few

drawbacks to using $PbWO_4$ crystals. One is the low light yield (LY), which is only about 100 photons per MeV for a 23 cm long crystal. This requires the use of a photodetector readout system with internal gain. Another disadvantage is the strong dependence of the light yield on temperature which requires temperature stability.

The energy, position, and time resolution of arrays of crystals have been thoroughly studied in beam tests with no magnetic field, no material upstream of the crystals, no radiation damage, and minimal variation in channel response. The energy resolution obtained for the central impact of electrons on a 3 x 3 crystal array consists of a stochastic term, a noise term, and a constant term:

$$\frac{\sigma_E}{E} = \frac{2.8\%}{\sqrt{E}} \bigoplus \frac{0.128GeV}{E} \bigoplus 0.3\%, \tag{1.5}$$

where E is measured in GeV. The constant term in the energy resolution is largely influenced by the non-uniformity of the light collection along the longitudinal axis. Material upstream of the ECAL can cause photon conversion and electron Bremsstrahlung, which can impact all terms in the energy resolution. The goal of the CMS was to achieve a constant term of below 1% [21].

1.3.4 Hadron Calorimeter

The Hadron Calorimeter (HCAL) [22] and ECAL are used to measure the direction and energy of hadronic jets as well as calculate the amount of missing transverse energy (missing E_T) for each occurrence. The need for an accurate missing E_T measurement necessitates the creation of a very hermetic system, whose design is limited by the strong magnetic field and compactness requirements. A sample calorimeter device based on brass absorber layers interspersed with active plastic scintillators has been constructed to meet these requirements. With incorporated wavelength-shifting fibres (WLS), the signal from active scintillators is read out and sent to hybrid photodiodes via clear fibre waveguides. Brass was chosen as the absorber material due to its low interaction length (λ_I) and lack of magnetic properties. Figure 1.8 displays a longitudinal perspective of the HCAL arrangement. The hadron calorimeter is a sampling calorimeter made of a copper alloy absorber and plastic scintillators, and it is located behind the tracker and the electromagnetic calorimeter as seen from the interaction point. The construction of the barrel hadron calorimeter (HB) is polygonal. It is constructed of two parts, each of which is made up of 18 wedges. Each wedge measures 20° inches in width and 4.33 meters in length in the z-direction. Each of the 17 slots, which are spaced at regular radial spacing, has a negative scintillator. HB can reach up to $|\eta|$ = 1.3 - 1.4. Between the outside extent of the electromagnetic calorimeter (R = 1.77 m) and the inner extent of the magnet coil (R= 2.95 m), HB is radially constrained. This limits how much material may be used to completely absorb the hadronic shower.

1.3.5 Magnetic System

The CMS magnet system [24] is created to take into account the exact measurements of momentum of charged particles and identification of the charge of high transverse momentum muons required at the LHC. The CMS magnet system consists of a superconducting solenoid magnet that generates a strong, uniform magnetic field inside the detector. The

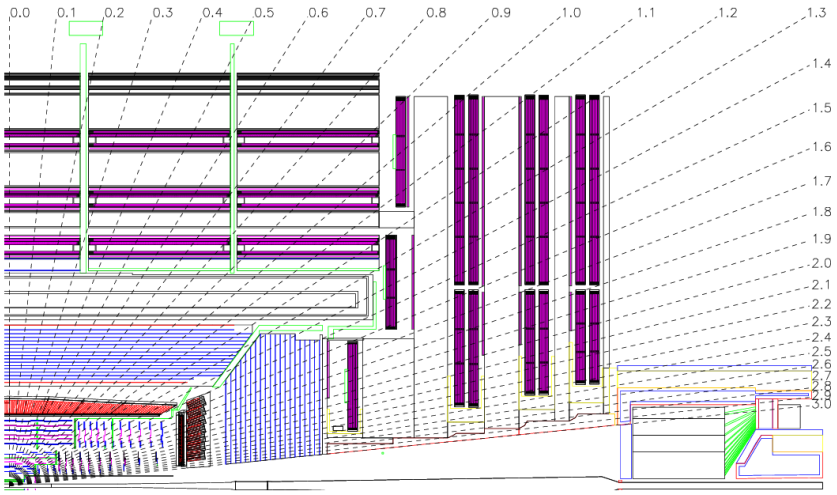


Figure 1.8: View of the CMS detector along its longitudinal axis with some fixed η lines [23].

solenoid magnet is made up of a cylindrical coil of superconducting wire, surrounded by a cryostat and various support structures. The magnet is capable of generating a field strength of about 3.8 T.

The CMS magnet system also includes several other magnets, such as trim coils and correction coils, which are used to fine-tune the magnetic field and correct for any distortions. These magnets are powered by a complex system of high-voltage power supplies and electrical current leads. The CMS magnet system is a critical component of the CMS experiment, as it helps to steer and focus the trajectories of charged particles as they pass through the detector. It is constructed of copper-wrapped NbTi cables, cooled to 4.5 K, and housed inside a 12,000-ton iron yoke that also houses the outside muon detection equipment. The iron yoke is used for the magnetic return flux. The iron yoke's 2 T residual field has sufficient bending power to carry out an effective p_T-based muon trigger selection in a region of $|\eta| < 2.4$ pseudorapidity.

1.4 The Muon System

Many of the intriguing physical processes anticipated at the LHC will exhibit final states that will contain large p_T muons. To offer accurate muon identification, performed with the muon system, located outside the superconducting coil (Figure 1.9), high resolution p_T measurements, and a reliable trigger capability, a robust and redundant muon spectrometer is required. High-energy muons are created in proton-proton collisions and are especially detected in the Muon System, a distant external set of subdetectors of the CMS experiment, after crossing both electromagnetic and hadronic calorimeters.

Three different types of gaseous detectors [25], each with a unique design to accommodate the radiation environment and magnetic field at various values of η, make up the experimental muon setup:

- 250 Drift Tube Chambers (DT) are utilized in the barrel region (with $|\eta| < 1.2$), where there is little track occupancy and a weak residual magnetic field;
- to deal with a larger particle flux and an uneven magnetic field, the endcaps ($0.8 <$

$|\eta| < 2.4$) are outfitted with 540 Cathode Strip Chambers (CSC) with quicker and radiation-resistant capabilities;

- due to their quick response, excellent time resolution, and low spatial resolution, 610 Resistive Plate Chambers (RPC) complement the DT and CSC in both regions up to $|\eta| < 2.1$ to ensure redundancy and improve trigger performances. This improves the precision in the muon trigger on the determination of the bunch crossing (BX) in which the muon has been created.

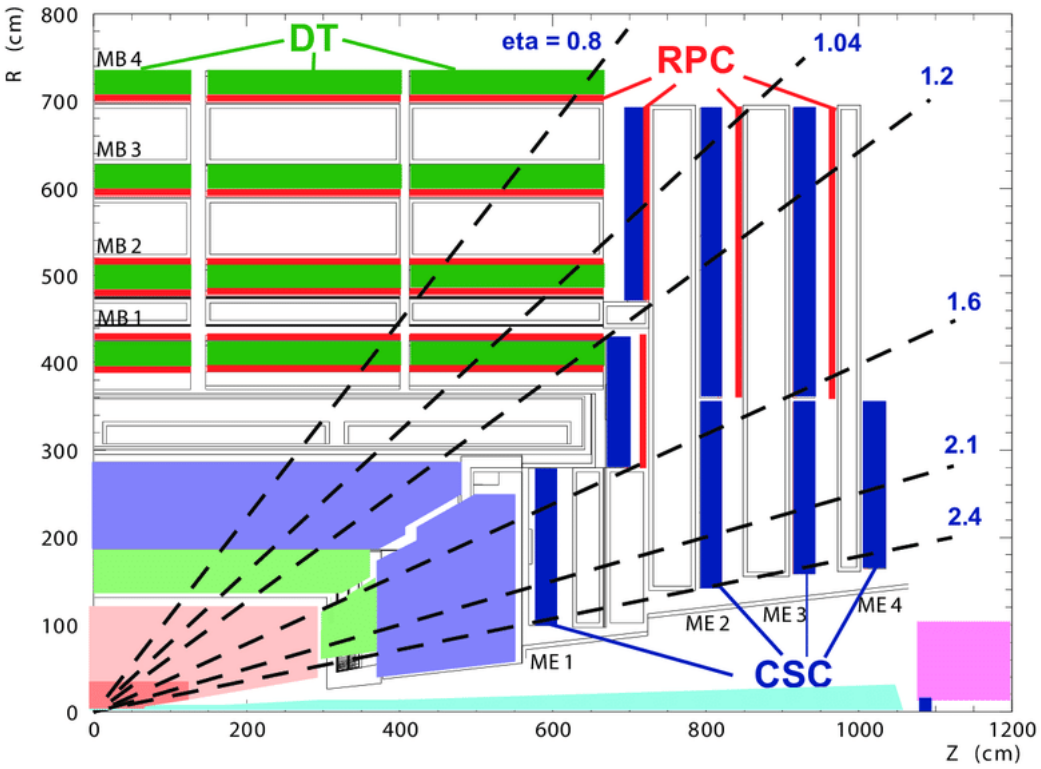


Figure 1.9: Layout of the CMS muon system in a longitudinal quarter view [26]. Only the inner ring of the ME4 chambers has been deployed for the CSC system, and the RPC system is only capable of $|\eta| < 1.6$ in the endcap.

Due to the repeated scattering that occurs when a muon crosses a calorimeter and the iron yoke of the muon portion, the tracker precision is most important for muons with $p_T < 200GeV$, although at higher p_T the combination of the two systems increases the overall resolution.

1.4.1 Drift tube system

The drift tube cell [27], scehmatic is shown in Figure 1.10), which measures 42 mm by 13 mm and has a stainless steel anode wire with a diameter of 50 m and a length ranging from 2 to 3 m, is the basic detector component of the DT muon system. Cells are arranged in layers between two parallel aluminium planes, which are separated from one another by "I"-shaped aluminium beams. Cathodes are electrically isolated strips of aluminium that have been deposited on both faces of each I-beam. The positive and negative voltages (usually +3600 V and -1200 V, respectively) of the anode wires and cathodes create the

electric field inside the cell volume. To provide additional field shaping and enhance the space-to-distance linearity over the cell, two additional positively biased (+1800V) strips are mounted on the aluminium planes on both inner surfaces in the centre of the cell. This distance of the traversing track to the wire is measured by the drift time of ionization electrons. A gas mixture of 85% CO2 and 15% Ar is used to fill the tubes, which has good quenching capabilities. About $55\mu m/ns$ is the obtained drift speed. This results in a maximum drift duration (half-cell drift distance) of ≈380 ns (or 15–16 BXs). The low predicted rate and the comparatively weak local magnetic field were the driving factors in the decision to use a drift chamber as the tracking detector in the barrel.

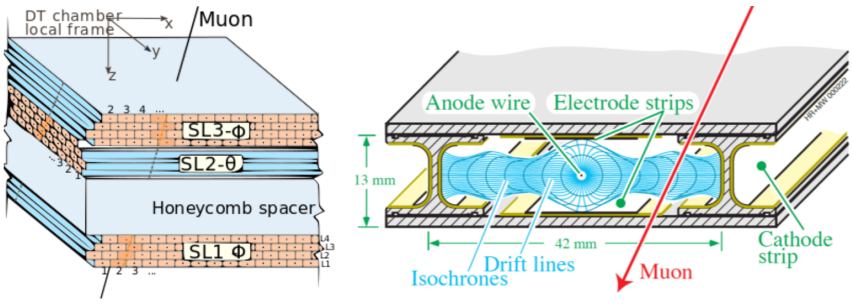


Figure 1.10: (*left*) a diagram of the DT chamber in the Muon System; (*right*) a drift tube cell [28].

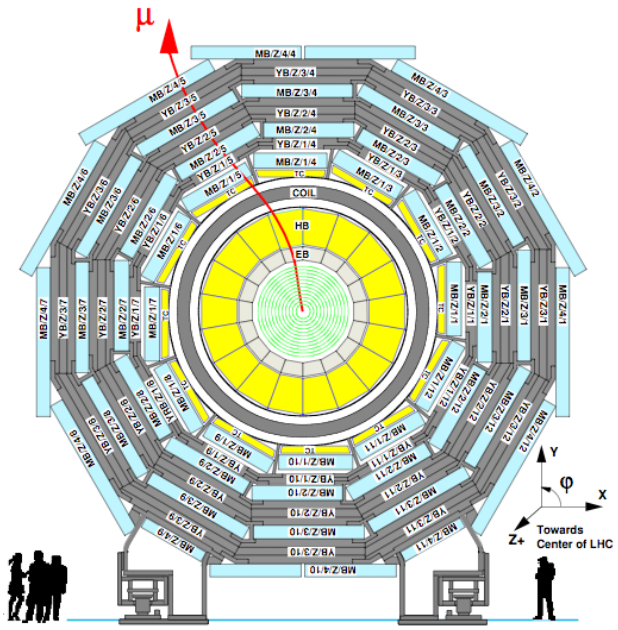


Figure 1.11: A cross-section of the CMS DT system. In the figure, station and sector numbers are displayed.

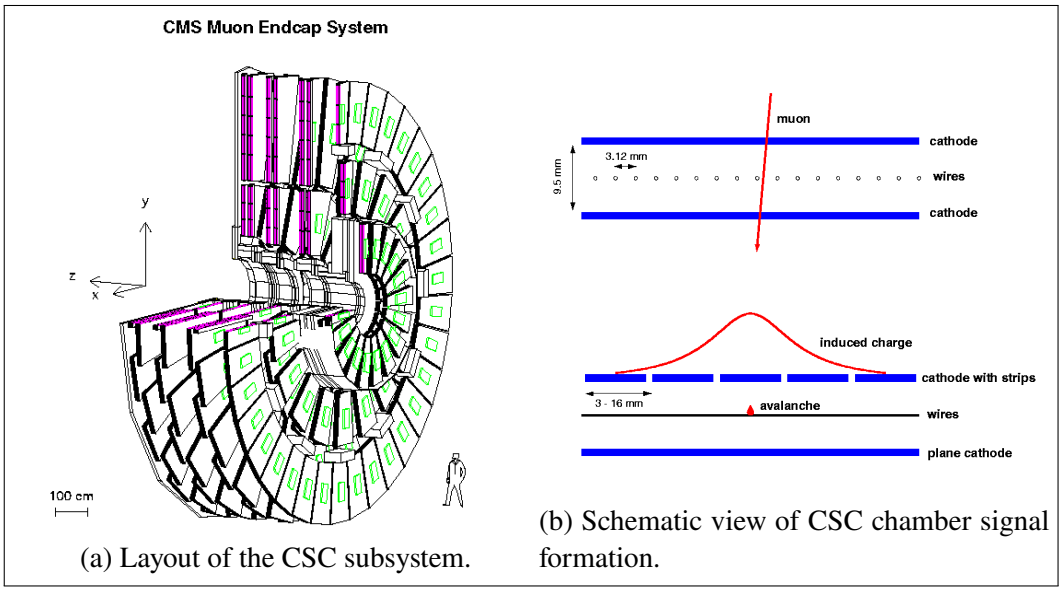
The DT system is divided into 12 azimuthal sectors (Figure 1.11), each covering 30°, and 5 wheels along the z-axis, each around 2.5 m wide. Within each wheel, drift tubes are organized in 4 concentric cylinders, known as stations, spaced variously from the contact point and interspersed with the iron of the yoke. Except for the outermost station, MB4, which has top and bottom sectors with two chambers each, each DT station has 12 chambers in each wheel, for a total of 14 chambers in that station. To maximize geometric

acceptability, each DT chamber is azimuthally staggered to the preceding inner one. To create three superlayers (SL), the DT layers inside a chamber are stacked, half-staggered, in groups of four.

Two of these superlayers (SL) measure the muon location in the bending plane r-ϕ, while the third super layer measures the position along the z coordinate. The MB4 station's outermost chambers, however, only have two phi superlayers installed. This means that there are 250 DT chambers in the entire CMS detector. While r can be calculated with a precision of ≈0.5 cm, the location resolution measured with the strips ranges from ≈ 70 mm for the innermost stations to ≈ 150 mm for the outermost ones.

1.4.2 The Cathode Strip Chambers

Drift tube detectors cannot be used to conduct measurements at high η values due to the high magnetic field and particle rate anticipated in the muon system endcaps. As a result, a Cathode Strip Chambers (CSC)-based system was chosen [29]. The CSC are gaseous trapezoidal Multiwire Proportional Chambers (MWPC), which are distinguished by a brief drift length that facilitates quick signal gathering. Both in the anode wire and on a collection of thinly divided cathode strips, data are gathered regarding the position of the arriving particle. Figure 1.12 illustrates the CSC layout. These chambers are set up to create four disks of concentric rings that are positioned in the space between the endcap iron yokes.



(a) Layout of the CSC subsystem.

(b) Schematic view of CSC chamber signal formation.

Figure 1.12: The Cathode Strip Chambers of the CMS endcap muon system.

Each chamber is made up of six layers of 9.5 mm thick anode wire arrays that are sandwiched between two planes of cathode strips with finely segmented edges to receive the ionization signal generated by the mixture of 30%/50%/20% Ar/CO2/CF4 gas. The strips are used to calculate the polar angle, while the wires provide information about the r coordinate. The anode wire in the first disk, which operates in a high magnetic field area, is slanted by 20° to account for the Lorentz drift effect. While r can be calculated with a precision of ≈ 0.5 cm, the location resolution measured with the strips ranges from ≈70 μm for the innermost stations to ≈150 μm for the outermost ones.

1.4.3 The Resistive Plates Chambers

Resistive Plates Chambers are employed in the barrel and endcaps, completing the DT and CSC systems, to ensure the muon spectrometer's durability and redundancy. RPC are gaseous detectors that have a coarse spatial resolution but are nonetheless capable of performing accurate time measurements that are on par with those offered by scintillators [30]. The muon trigger mechanism will receive a precise BX identification as a result.

The 90%/10% mixture of freon ($C_2H_2F_4$) and isobutane (C_4H_{10}) is ionized by crossing particles in double-gap RPC chambers made up of four bakelite planes generating two 2 mm gaps, as shown in Figure 1.13, and the electrons are multiplied in avalanche mode. The 9.5 kV electrodes are made of a graphite coating, and the centre of the chamber is fitted with insulated aluminium strips that are utilized to gather the signal produced by the electronic avalanche brought on by the crossing particles. To boost the signal imparted to them, the double gap chamber design decision was used. Along the beam axis, the strips in the barrel are rectangularly segmented (12.1 to 41 cm wide and 80 to 120 cm long), whilst the endcaps are fitted with trapezoidal-shaped strips that roughly span the range $\Delta\phi = 5 - 6°$, $\Delta\eta = 0.1$. Except for the restriction imposed by the strip length, no measurement is feasible in the η coordinate. The detector operates in *avalanche* mode rather than the more conventional *streamer* mode to support greater speeds, but due to the reduced gas multiplication, improved electronic multiplication is needed.

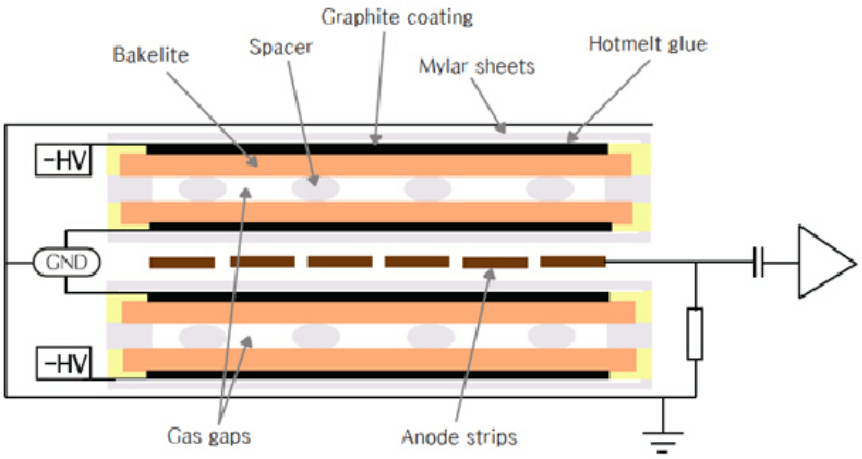


Figure 1.13: Cross sectional view of a CMS double gap RPC [31].

Two RPC stations are attached to each side of the two innermost DT chambers of a sector in the barrel region of the system, whereas only one RPC station is attached to the inner side of the third and fourth DT chambers. The system is laid out according to the DT segmentation. By detecting even low p_T muons before they stop in the iron yoke, this technique guarantees to increase the low p_T range of the trigger mechanism in the barrel.

1.5 Trigger and Data Acquisition

At the LHC, proton bunches collide at a rate of 40 MHz. Considering that, each event in the two general-purpose experiments ATLAS and CMS is around 1 MB in size, a trigger system is needed to select the most interesting events for further analysis. To reduce the

event rate to a manageable level for the final storage, the trigger must achieve a reduction rate of 10^7, allowing for a maximum event rate of ≈1 kHz. The trigger algorithms must be carefully tuned to be sensitive to a wide range of physical processes, including ones with a low probability, like: $Z \to ll$ and $W \to l\nu$, but simultaneously rejecting a large number of backgrounds. Because procedures like QCD have substantially higher cross sections and might potentially saturate selections based on simple high p_T lepton identification, this presents a particularly difficult problem.

The high frequency of 40 MHz at which the BX occurs requires the development of a system that can make a selection or rejection decision every 25 ns. A pipelined trigger architecture with many layers of increasing complexity has been devised since the time frame is too short to collect and process all the essential data from all the subdetectors in one go. The CMS trigger system [32] has two levels of event selection to identify those that may be of potential interest for physics studies:

- the first level (L1T) is implemented in custom hardware and selects events that contain candidate objects such as muon-like ionization deposits, electron- or photon-like energy clusters, τ leptons, missing transverse energy, or jets. The scalar sum of jet transverse momenta (HT) can be used to select events with potentially large momentum transfers.

- The final event selection for the CMS trigger system is based on a programmable menu that uses up to 128 algorithms to analyze the candidate objects identified in the first level of selection. This process, known as the high-level trigger (HLT), is implemented in software and further refines the purity of the selected objects. The thresholds of the first level are adjusted during data taking to ensure that the output rate does not exceed 100 kHz, the upper limit imposed by the CMS readout electronics. The HLT has an average output rate of 400 Hz for offline storage, which can be adjusted through the use of prescaling to reduce the number of events that pass the selection criteria of specific algorithms.

In addition to collecting collision data, the trigger and data acquisition systems also record information for the monitoring of the detector.

1.5.1 The Level-1 Trigger System

The Level-1 trigger [33] uses uniquely designed programmable hardware, such as field programmable gate arrays (FPGAs), application specific integrated circuits (ASICs), and programmable lookup tables (LUTs), to quickly accept or reject events at a rate of 40 MHz. In a pipeline mode, each processing step must be completed in less than 25 nanoseconds, with pipeline buffers having a maximum length of 128 BX (3.2 μs). When the signal propagation time and the latencies of the subdetectors are taken into consideration, the L1T's effective decision operation time is, nevertheless, less than 2 μs.

The Muon Trigger, the Calorimeter Trigger, and the Global Trigger are the three primary subsystems that make up the Level-1 Trigger of the CMS detector. The first two systems gather data from the calorimeters and muon spectrometers, but they are not in charge of accepting or rejecting events on their own. The Muon Trigger and Calorimeter Trigger subsystems identify and prioritize various types of trigger objects, such as electrons/photons, jets, and muons, and then pass the top four candidates of each

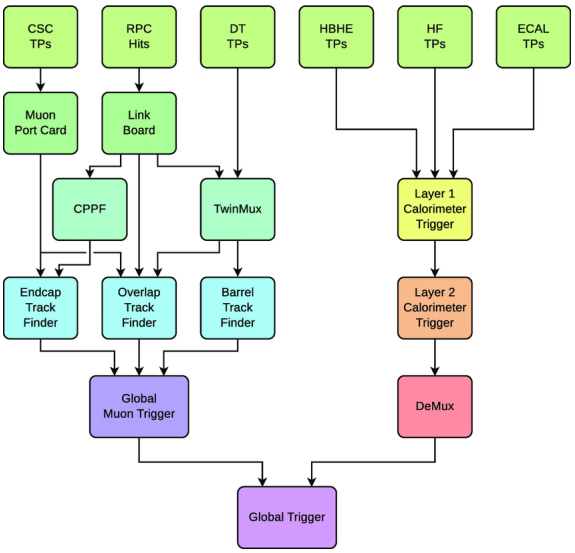


Figure 1.14: Diagram of the upgraded CMS Level-1 trigger system during Run 2 [34].

type to the Global Trigger. The Global Trigger then makes the final decision on whether to accept or reject the event, as shown in Figure 1.14.

1.5.2 The High-Level Trigger and DAQ

The L1 Trigger's event rate must be further decreased to roughly 1 kHz by the CMS High Level Trigger (HLT) to meet the storage system's requirements. The HLT uses a farm of commercial processors to do an analysis akin to offline event reconstruction to achieve this [35].

The Front End System (FES) first keeps the data produced by the Level-1 Trigger in a 40 MHz buffer. The Front End Drivers (FEDs) then transfer the data to the Front End Readout Links (FRLs), which can receive information from two different FEDs. The event fragments received by the FRLs are assembled into full events by the Event Builder system, which has two main purposes: to transport the data from underground to the CMS surface buildings and to initiate the reconstruction phase. After the event has been assembled, it is transmitted to the event filter, where HLT algorithms and operations for assessing the quality of the data are carried out. Thereafter, the filtered data are split up into many online streams, the content of which depends on the trigger setup, and they are delivered to a local storage system before being forwarded to the CERN mass storage infrastructure.

Chapter 2

Machine and Deep Learning

Machine and Deep learning have become increasingly relevant techniques in High Energy Physics (HEP) research. These methods allow physicists to analyze and interpret large amounts of data generated by experiments such as CMS and ATLAS at the LHC.

Machine learning involves training algorithms on data to make predictions or decisions. Deep learning is a type of machine learning that uses neural networks, which are inspired by the structure and function of the human brain, to recognize patterns and make decisions based on input data. Algorithms based on deep learning have been successful in a variety of applications, including image and speech recognition.

Deep learning and machine learning can be used in HEP to analyze and interpret experimental data, identify elusive or rare particles, forecast the results of upcoming experiments, or locate patterns that are out-of-the-ordinary or abnormal, such as the Anomaly Detection task.

This chapter introduces the concepts of machine learning and deep learning and discusses some of their applications in high energy physics research, with a particular focus on the Variational Autoencoder model, used extensively in the analysis carried out in the following chapter.

2.1 Introduction to Machine Learning

Machine Learning (ML) [36] is a branch of artificial intelligence that is focused on developing algorithms that can learn from data automatically. This allows artificially intelligent agents to recognize objects and predict the behaviour of their environment to make informed decisions. ML techniques are typically more focused on making **predictions**, rather than estimating values. They are also often used to solve complex, high-dimensional problems that are beyond the scope of traditional statistical methods. For example, ML techniques might be used to predict the interference pattern that would be observed in an interferometry experiment under different conditions, based on data from previous experiments [37].

Despite their differences, estimation and prediction problems can be approached using a similar framework. In both cases, we choose some observable quantity x of the system under study, such as an interference pattern that is related to some parameters θ, e.g. the speed of light, and a model $p(x|\theta)$ that describes the probability of observing x given θ. We then perform an experiment and use the data X we collect to fit the model to the data. This typically involves finding the parameter values that provide the best explanation for the data. In the case of least squares estimation, the parameters that are chosen maximize the probability of observing the data (i.e., $\hat{\theta} = \arg\max_\theta p(X|\theta)$). *Estimation* problems are concerned with the accuracy of $\hat{\theta}$ these estimated parameter values, while *prediction* problems are concerned with the ability of the model to make accurate predictions about new observations, (i.e. the accuracy of $p(x|\hat{\theta})$). These goals can sometimes lead to different approaches to solving the problem.

In the past thirty years, we have seen a significant increase in our ability to collect, process, and analyze large amounts of data, known as *Big Data*. This growth has been fueled by the rapid advancement of technology, particularly the increasing power and capacity of computers, as described by Moore's law. Specialized computing systems, such as those using graphics processing units (GPUs), are making it possible to perform large-scale computation at lower costs, indicating that the big data revolution is likely to continue in the future. Along with the advancements in computing power, new methods for analyzing and extracting insights from large data sets have also emerged. These techniques often incorporate concepts from various fields, including statistics, neuroscience, computer science, and physics. In particular, modern ML approaches often prioritize practical, real-world results and intuition, rather than relying on more formal, theoretical approaches that are common in disciplines such as statistics, computer science, and mathematics.

Physicists have both the opportunity to benefit from and contribute to the development of machine learning. Many ML techniques, including Monte Carlo methods, simulated annealing, and variational methods, have roots in physics. In addition, the concept of *energy-based models* [38], which is inspired by statistical physics, is central to many deep learning methods. As a result, physicists are well-equipped to engage with and contribute to the field of ML.

2.1.1 Problem Formulation in Machine Learning

In many cases, machine learning problems begin with similar elements. One of the essential elements is a dataset $\mathcal{D} = (X, y)$, where X denotes an input matrix of independent variables and y is an output variable vector. The model is denoted as $f(x;\theta)$, where $f : x \to y$ is a function that maps input variables to output variables through a set of parameters θ. In summary, the model is used to predict outputs from a vector of input variables. To evaluate the performance of the model, a cost or loss function $\mathcal{C}(y, f(X;\theta))$ is used. The goal is to find the set of parameters θ that minimizes this cost function, effectively "fitting" the model to the observations y.

ML involves a structured process for developing models that can effectively predict outcomes. A key step in this process is the initial partitioning of the dataset $\mathcal{D}$, into two distinct groups: the training set $\mathcal{D}_{train}$ and the test (validation) set $\mathcal{D}_{test}$. In most cases, the data is divided into the training set, which comprises a significant proportion of the data (e.g. 85%), and the remaining portion is allocated to the test set. The model is then fitted by using only the training set data to minimize the cost function, so

the minimization expressed as $\hat{\theta} = \arg\min_\theta\{\mathcal{C}(y_{train}, f(X_{train}; \theta))\}$. In the final step, the model's performance is evaluated by calculating the cost function using the test set $\mathcal{C}(y_{test}, f(X_{test}; \hat{\theta}))$. The cost function value for the optimal model fit on the training set is referred to as the in-sample error, represented as $E_{in} = \mathcal{C}(y_{train}, f(X_{train}; \theta))$. The cost function value on the test set, on the other hand, is known as the out-of-sample error, represented as $E_{out} = \mathcal{C}(y_{test}, f(X_{test}; \theta))$.

An important insight that can be derived from the analysis is that the out-of-sample error typically exceeds the in-sample error, meaning $E_{out} \geq E_{in}$. Dividing the data into mutually exclusive training and test sets offers an unbiased estimate of the model's predictive capabilities, a method commonly known as cross-validation in literature. In numerous applications of classical statistics, the goal is to estimate the value of some unknown model parameters using a mathematical model that is frequently considered to be correct. In contrast, problems in ML often involve understanding complex systems where the precise mathematical model describing the system is unknown. Therefore, it is common for ML researchers to consider multiple potential models, and compare them using the out-of-sample error (E_{out}). The model that minimizes this error is typically chosen as the optimal model (model selection). It is important to recognize that once the best model is selected by evaluating its performance on E_{out}, the actual performance of the chosen model in real-world scenarios is likely to be slightly poorer as the test data was used in the training procedure.

2.1.2 Overfitting and Underfitting

One of the main challenges in machine learning is the ability to accurately predict outcomes for new, unseen data, rather than just for the data that was used to train the model. This is often referred to as generalization, and the model needs to be able to perform well on a wide range of inputs. To achieve good generalization, it is necessary to carefully design and train the model, choosing appropriate algorithms and parameters, and ensuring that the model has not overfit the training data.

Overfitting occurs when a model is too complex and has learned patterns that are specific to the training data, but may not generalize to other data. This can happen when the model has too many parameters, or when the training data is not representative of the broader population. As a result, the model may perform well on the training data, but it will not be able to generalize to new, unseen data, and its performance will suffer. If the model performs well on the training data but poorly on the validation data, it is likely overfitting. To address overfitting, several approaches can be effective:

- reduces the complexity of the model by using fewer parameters or by using regularization techniques, which constrain the model to prevent it from learning overly complex patterns;
- increases the size of the training dataset, which can help the model learn more general patterns.

Underfitting, on the other hand, occurs when a model is too simple and is not able to capture the underlying patterns in the data. This can happen when the model has too few parameters, or when the training data is not sufficient to learn the desired patterns. In this case, the model will perform poorly on both the training data and new, unseen data. To identify underfitting, it is again useful to evaluate the model's performance on a validation

dataset. If the model performs poorly on both the training and validation data, it is likely underfitting. To address underfitting, several approaches can be effective:

- increases the complexity of the model by using more parameters or by using more flexible model architectures;
- increases the size of the training dataset, which can provide the model with more examples to learn from.

2.1.3 Basics of Statistical Learning

This section provides a general overview and explores the concept of learning, specifically the conditions under which it can be achieved. Let's consider an unknown function, represented as $y = f(x)$. To learn this function, we define a set of hypotheses $\mathcal{H}$, which includes all the functions we are willing to consider. The specific functions included in this set are based on our understanding and assumptions about the problem at hand. Our objective is to select the best approximation of $f(x)$ from the hypothesis set $\mathcal{H}$, by finding a function h that closely matches f in a mathematical sense. If we are successful in doing this, we can say that we have *learned* the function $f(x)$. After selecting an error function E, that is appropriate for the problem at hand, we can evaluate the performance of a model using two distinct measures: the in-sample error, E_{in}, and the out-of-sample or generalization error, E_{out} (as previously discussed in section 2.1.1). To illustrate some of the core concepts in statistical learning theory [39] — which have a considerable influence on how we view and approach machine learning — this section will provide two schematics.

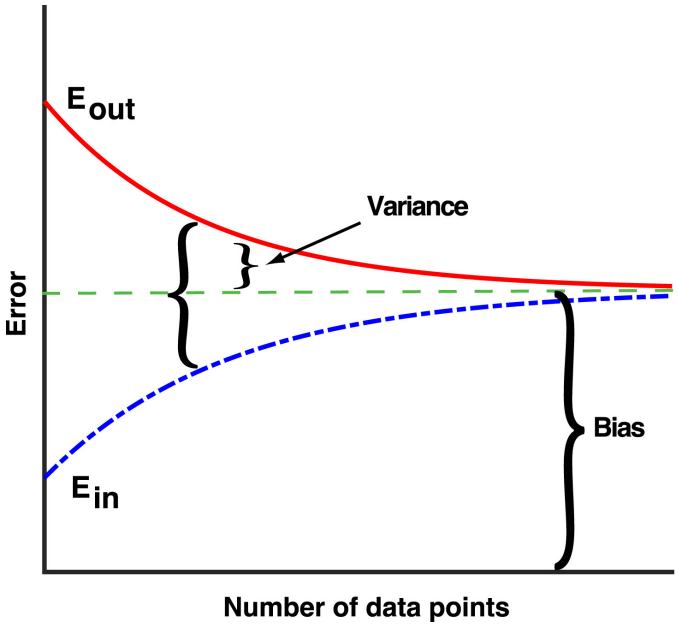


Figure 2.1: An illustration showing how typical in-sample and out-of-sample error changes with training set size [37].

The first schematic shown in Figure 2.1 illustrates the typical relationship between the out-of-sample error E_{out}, and in-sample error E_{in}, as the amount of training data increases. The graph is based on the assumption that the true data is generated from a complex distribution, making it impossible to exactly learn the function $f(x)$. Therefore,

as the number of data points increases, the in-sample error will rise, as our models cannot perfectly learn the true function that we are trying to approximate, after a brief initial drop that is not depicted in the graph. In contrast, as the number of data points increases, the out-of-sample error will decrease. As the size of the training set grows, the noise generated by sampling diminishes, and the training set becomes more representative of the true data distribution. As a result, as the number of data points approaches infinity, the in-sample and out-of-sample errors will converge to the same value, referred to as the *bias* of the model. The bias represents the best that our model can achieve given an infinite amount of training data to overcome sampling noise. The bias is a characteristic of the type of functions or model class that we are using to approximate $f(x)$. However, in practice, we do not have an infinite amount of data. Therefore, to achieve the best predictive power, it is better to minimize the E_{out}, rather than the bias. As shown in Figure 2.1, E_{out} can be broken down into bias and variance. Bias measures how well we can perform in an ideal scenario with infinite data, while variance measures the typical errors that arise due to sampling noise when working with a finite training set. In Figure 2.1, the last value shown is the gap between the generalization error and the training error. It gauges how closely the in-sample error approximates the out-of-sample error, and it quantifies how much worse our model's performance would be on new data compared to the training data. Therefore, this difference represents the distinction between fitting the data and making predictions, it is a measure of overfitting and underfitting.

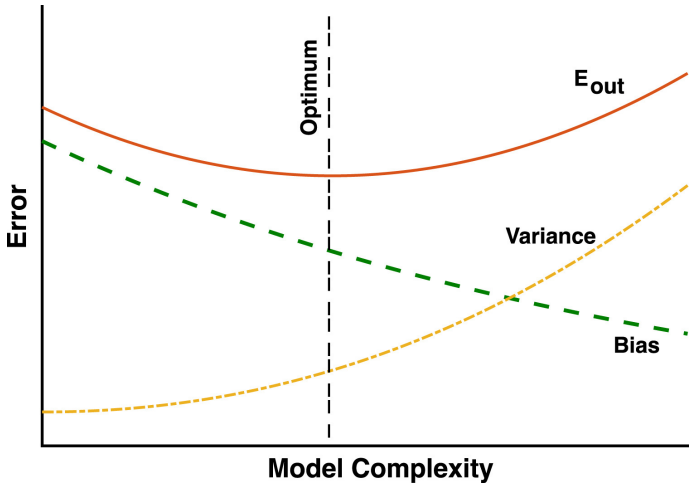


Figure 2.2: The trade-off between bias and variance. As the complexity of a model increases, it can more closely match the underlying relation, but there is also a greater risk of increasing the variance, leading to overfitting. To minimize the total prediction error (which is the sum of bias and variance), it is necessary to balance the bias and the variance by considering the trade-off between these two factors [37].

The second diagram in Figure 2.2, illustrates the out-of-sample or test error E_{out} as a function of model complexity. In many cases, model complexity is related to the number of parameters used to approximate the true function $f(x)$. When considering a fixed-size training dataset, E_{out} is generally not a monotonically increasing or decreasing function of model complexity, instead it is minimized for models with an intermediate complexity. This is because, although using a more complex model always decreases bias, at a certain point, the model becomes overly complex with the amount of training data, and the generalization error increases significantly due to high variance. To minimize E_{out} and enhance predictive power, it may be more beneficial to use a model with a higher bias

and lower variance rather than a model with a lower bias but higher variance. This crucial idea is referred to as the *bias-variance trade-off*.

2.1.4 Gradient Descent

As it is described so far, most ML tasks begin with similar components: a dataset X, a model $g(\theta)$ which is a function of parameters θ, and a cost function $\mathcal{C}(y_{test}, f(X_{test}; \hat{\theta}))$ that measures how well the model $g(\theta)$ explains the observations in X. The goal is to find the values of θ that minimize the cost function, to optimize the performance of the model.

It is now examined the most powerful and widely used methods for minimizing the cost function - gradient descent and its variants. These methods are based on the principle of iteratively adjusting the parameters θ in the direction where the gradient of the cost function is large and negative. This training procedure aims to guide the parameters towards a local minimum of the cost function. However, in practice, Gradient Descent (GD) can be challenging and a range of techniques have been developed by the optimization and machine learning communities to enhance the performance of these algorithms. The main reason why training a machine learning model is difficult is that the cost functions used are often complex, rugged, non-convex, high-dimensional functions with multiple local minima. Additionally, the true function to be minimized is rarely known, instead, it must be estimated directly from the data. In modern applications, the size of the dataset and the number of parameters to be fit can be immense (millions of parameters and examples).

The basic idea behind gradient descent is that by moving in the direction of the negative gradient, the algorithm will be able to reduce the value of the loss function and, therefore, find better solutions. The algorithm continues this process until it reaches a local minimum of the cost function, where the gradient is zero. The function $E(\theta)$ represents the cost or error (or loss) of the model. The cost function can typically be written as a sum over all data points,

$$E(\theta) = \sum_{i=i}^{n} e_i(x_i, \theta). \tag{2.1}$$

and in the gradient descent algorithm, the model's parameters are updated in the direction that reduces this cost. Initialize the parameters θ to some value θ_0 and iteratively update the parameters according to the equation

$$\begin{aligned} v_t &= \eta \nabla_\theta E(\theta_t), \\ \theta_{t+1} &= \theta_t - v_t \end{aligned} \tag{2.2}$$

where $\nabla_\theta E(\theta)$ is the gradient of $E(\theta)$ w.r.t. θ and we have introduced a *learning rate* η, that controls how big a step we should take in the direction of the gradient time step t. It is evident that with a small enough value for the learning rate η, this method will converge to a *local minimum* of the cost function in all directions. However, using a small learning rate η comes with a high computational cost. The smaller the learning rate, the more steps are needed to reach the local minimum. On the other hand, if the learning rate is too high, the algorithm may overshoot the minimum and become unstable (it either oscillates or even moves away from the minimum). This is shown in Figure 2.3.

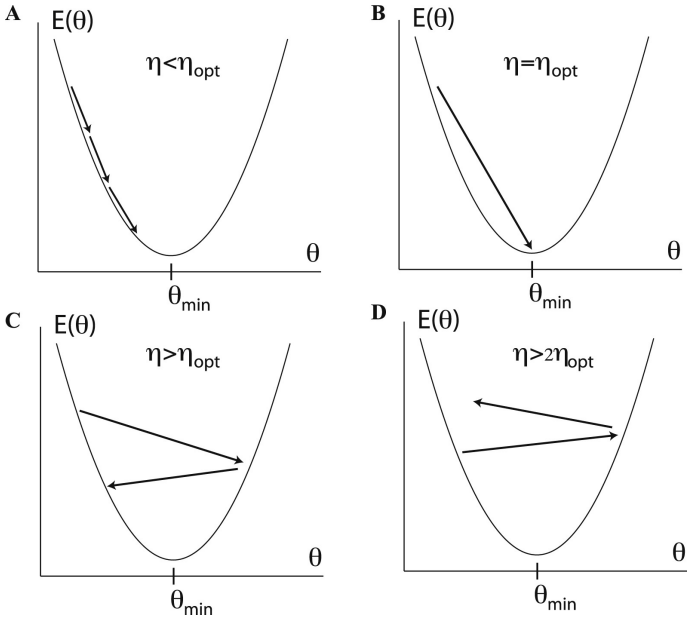


Figure 2.3: The impact of the learning rate on the convergence of the algorithm can be observed through its effect on a one-dimensional quadratic potential. By analyzing the relationship between the learning rate η and the optimal rate $\eta_{opt} = |\partial_\theta^2 E(\theta)|^{-1}$, it has been shown that GD can exhibit four distinct behaviours depending on the chosen value of the learning rate. (a) for $\eta < \eta_{opt}$, GD converges. (b) For $\eta = \eta_{opt}$, GD converges in a single step. (c) For $\eta_{opt} < \eta < 2\eta_{opt}$, GD oscillates around the minima. (d) For $\eta > 2\eta_{opt}$, GD moves away from the minima [37].

There are different variants of gradient descent algorithms [40], such as batch gradient descent, stochastic gradient descent, and mini-batch gradient descent. Batch gradient descent computes the gradient for the entire training dataset at each iteration, which can be computationally expensive for large datasets. Stochastic gradient descent [41] updates the parameters after each training sample, which can lead to faster convergence, but also increase the variance of the parameter updates. Mini-batch gradient descent [42] combines the advantages of both batch gradient descent and stochastic gradient descent, by updating the parameters based on a small subset of the training data.

In addition to the traditional gradient descent, several optimization methods perform better in many cases such as Adam [43], Adagrad [44], Adadelta [45], RMSprop [46] and so on. These are optimization algorithms that are more robust to the choice of hyperparameters and have been used in several tasks of deep learning with great results. In DL, the cost function is typically a complex non-convex function with many local minima. Therefore, the choice of initial parameters, the learning rate and the optimization algorithm used are important factors to converge to a good solution.

Adam Optimization Algorithm

Adam (Adaptive Moment Estimation) is an optimization algorithm used to update the parameters of a model during training. Adam combines the benefits of the gradient descent algorithm with the flexibility of adaptive learning rates for different parameters by keeping a running average of both the first and second moment of the gradient (i.e. $m_t = \mathbb{E}[g_t]$ and $m_t = \mathbb{E}[g_t^2]$). This allows Adam to adjust the learning rate in a more

effective way than other optimization algorithms. Additionally, Adam also performs bias correction to account for the fact that the first two moments of the gradient are being estimated using a running average. The specific update rule for Adam is given by the equation

$$\begin{aligned} g_t &= \nabla_\theta E(\theta) \\ m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ s_t &= \beta_2 s_{t-1} + (1-\beta_2) g_t^2 \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{s}_t &= \frac{s_t}{1-\beta_2^t} \\ \theta_{t+1} &= \theta_t - \eta_t \frac{\hat{m}_t}{\sqrt{\hat{s}_t} + \epsilon}, \end{aligned} \tag{2.3}$$

where β_1, β_2 are decay rates for the first and second-moment estimates, typically taken to be 0.9 and 0.99, and $(\beta_j)^t$ denotes β_j to the power t. η is the learning rate typically chosen to be 10^{-3}, m_t and s_t are the moving averages of the first and second gradients. $\hat{m}_t$ and $\hat{v}_t$ are the bias-corrected versions of m_t and v_t, respectively. $\epsilon \sim 10^{-8}$ is a small constant added to prevent division by zero. Adam typically requires less fine-tuning of the learning rate than the traditional gradient descent algorithm, and it has been used in various deep learning tasks with great results.

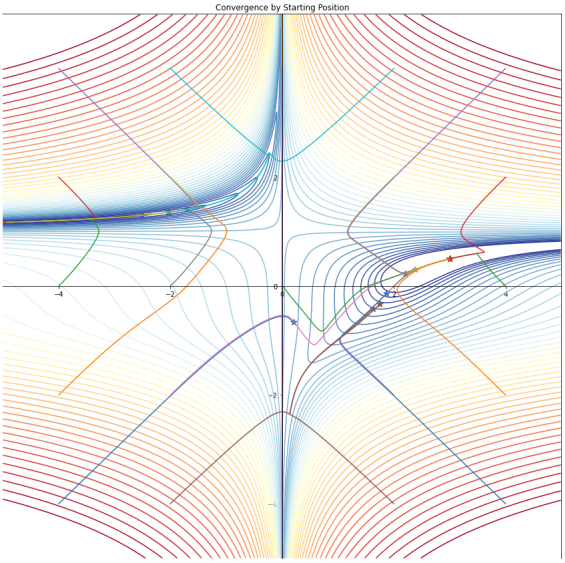


Figure 2.4: Adam and its generalization for Beale's function. Trajectory from $\beta_1 = 0.9$ $\beta_2 = 0.99$.

To better understand the performance of the Adam optimization method being considered, it can be useful to visualize it using a test function like Beale's function, the graphical result of the optimization algorithm is shown in Figure 2.4. The Beale's function is defined as:

$$f(x,y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2. \tag{2.4}$$

It has a global minimum at $(x, y) = (3, 0.5)$, but its landscape also has a deep and narrow ravine along $y = 1$. This makes it difficult to optimize, as it can lead to getting stuck in local minima. The performance of the optimization method can be visualized by observing the trajectory of the optimization process on Beale's function surface. In this case, it has been observed that the Adam optimization method performs better and faster than other optimization methods when it comes to converging to the global minimum of Beale's function.

AdamW Optimization Algorithm

AdamW [47] is a variation of the Adam optimization algorithm that separates the weight decay term from the gradient update. To see this, L_2 regularization in Adam is usually implemented with the below modification where $\lambda \in \mathbb{R}$ is the rate of the weight decay at time t:

$$g_t = \nabla_\theta E(\theta) + \lambda\theta_t, \tag{2.5}$$

this term pulls the weights towards zero, which helps to regularize the model and prevent overfitting during training. AdamW adjusts the weight decay term to appear in the gradient update:

$$\theta_{t+1} = \theta_t - \eta_t \left(\frac{\alpha \hat{m}_t}{\sqrt{\hat{s}_t} + \epsilon} + \lambda\theta_{t-1} \right) \tag{2.6}$$

where α typically taken to be 0.001. It has been decoupled weight decay and loss-based gradient updates in Adam as shown in the update rule 2.6; this gives rise to our variant of Adam with *decoupled weight decay* (AdamW). This modification to the typical implementation of weight decay in AdamW allows for a simpler hyperparameter search by separating the settings of weight decay and learning rate. Additionally, using this method has been shown to result in better training loss and generalization error as it regularizes variables with large gradients more effectively than traditional L_2 regularization.

2.1.5 Supervised Learning

Supervised learning is a type of ML where the goal is to learn a mapping from input variables (or features) to output variables (or labels), based on a labelled training dataset. In other words, the model is trained on a dataset that contains both the input and the correct output, so that it can learn to predict the output for new, unseen data.

In general, the training process consists of three main steps:

1. **data preprocessing**: This step involves cleaning, transforming and preparing the data for the model.
2. **Model training**: This step involves using a specific algorithm to find the optimal parameters of the model that minimize a predefined cost function, based on the training dataset.
3. **Model evaluation**: This step involves assessing the performance of the model on a separate test dataset, to estimate its ability to generalize to new, unseen data.

There are several types of supervised learning methods, some common ones include:

- **linear regression**: is used to model the relationship between a continuous output variable and one or more input variables. It is a linear approach that finds the best fit line or hyperplane that minimizes the sum of squared errors.
- **Logistic regression**: Logistic regression is used to model the relationship between a binary output variable and one or more input variables. It is a generalized linear model that finds the best fit line that separates the data into two classes and estimates the probability of an instance belonging to each class.
- **Decision trees** and **Random Forests** [48]: Decision Trees are used to model the relationship between a categorical or continuous output variable and one or more input variables. It is a non-parametric approach that creates a tree-like model by recursively partitioning the data into subsets based on the values of the input variables. Random Forest is an ensemble approach to decision trees which combines multiple decision trees to improve the overall performance of the model.
- **Support Vector Machine** [49]: It is a linear model for classification and regression tasks, it finds the hyperplane in a high- or infinite-dimensional space that maximally separates the different classes.
- **Neural Networks**: Neural Networks are a set of algorithms that try to recognize patterns in data, inspired by the structure and function of the brain. They consist of layers of interconnected nodes, or artificial neurons, that are trained to learn the mapping from inputs to outputs.

In High Energy Physics (HEP), supervised learning is used in several tasks such as classification, regression, and anomaly detection. For example, in particle physics, supervised learning can be used to classify events based on the signals observed in the detector. This is important because the signals are often very weak, and separating signals from the background can be difficult. Supervised learning can also be used to identify new particles, or to determine the properties of known particles. In such cases, supervised learning can be used to analyze large amounts of data quickly, and to identify patterns that are not easily visible to the human eye. Another example is in the detector's Calibration, in HEP detectors, the energy and position measurements are affected by various physical processes, and a correction has to be applied to the measured values to obtain the true values. Supervised learning can be used to learn the relation between the measured and true values and to correct the data in real time.

2.1.6 Unsupervised Learning

Unsupervised learning is a type of machine learning where the goal is to find patterns or features in the input data without any labelled output data. The algorithms learn from the data itself, without the guidance of a known target or output variable.

There are several types of unsupervised learning, some common ones include:

- **Clustering**: the goal is to divide the input data into groups or clusters, where each cluster is defined by some similarity metric. k-means and hierarchical clustering are popular examples of clustering methods.

- **Dimensionality reduction**: the goal is to reduce the number of features or dimensions of the input data while preserving as much information as possible. Principal component analysis (PCA) and t-distributed stochastic neighbour embedding (t-SNE) are popular examples of dimensionality reduction methods.
- **Density estimation**: is a method in statistics and machine learning used to estimate the probability density function of a given random variable. One common technique for density estimation is *Gaussian kernel density estimation*, which involves placing a Gaussian kernel function at each data point and then taking the average of all the kernels to obtain an estimate of the underlying density. Another technique is *Normalizing flows* [50], which is a method of density estimation that involves transforming a simple base distribution into a more complex target distribution through a series of invertible and differentiable transformations. It is particularly useful for modelling complex, high-dimensional distributions.
- **Anomaly detection**: the goal is to identify data points that deviate significantly from the rest of the data. One-class SVM [51] and Isolation Forest [52] are popular examples of anomaly detection methods.
- **Autoencoder**: is a type of unsupervised neural network which is designed to learn the feature representation of the data.
- **Generative models**: the goal is to learn a model that can generate new samples from the input data distribution, such as Variational Autoencoder (VAE) [53] and Generative Adversarial Networks (GAN) [54]. An alternative generative model to GANs is diffusion probabilistic models, based on the idea of approximating a target density by iteratively applying a series of simple transformations to a simple base distribution. The model learns to transform the base distribution into the target distribution by adjusting the parameters of the transformations through optimization. The diffusion process is typically represented by a normalizing flow, which is a series of invertible, differentiable transformations that allow for efficient computation of likelihoods and sampling from the model. Some examples of normalizing flows used in diffusion models [55] include RealNVP [56], Glow [57], and WaveGlow [58].

In HEP, unsupervised learning methods are often used to extract useful information from large, complex datasets, and to identify patterns that are not easily visible to the human eye. In the case of particle detectors, unsupervised learning can be used in the calibration. For example, one can use the autoencoder to learn a compact representation of the detector's response, so that it can be used to perform the correction in real time.

2.1.7 Self-Supervised Learning

Self-supervised learning is a ML technique in which a model learns from a dataset without the need for explicit labels or supervision. The model is trained to predict, generate, or reconstruct some aspect of the input data, using its structure or other learned features as a guide.

A common example of self-supervised learning is using a model to predict missing words in a sentence, also known as a language model. Another example is using a model to predict the next frame in a video, known as a video prediction model. In HEP, one example

of self-supervised learning is using a deep learning model to identify particle tracks in detector images. In this case, the model is trained on a dataset of images of detector hits, to learn to recognize the patterns of hits that correspond to particle tracks. The model is not given explicit labels indicating which hits belong to a particle track but instead must learn to identify the tracks by analyzing the patterns in the data. Another example is using a deep learning model to perform energy regression on calorimeter images. The model takes images of the calorimeter response and predicts the energy deposit by the particle, in this scenario labels are not provided to the model, it must learn this relationship through its structure. Self-supervised learning has the potential to be applied to many other problems in HEP, such as particle identification, event reconstruction, and anomaly detection [59], to name a few.

There are several techniques used in self-supervised learning, some of the most common include [60, 61]:

- **Contrastive Learning**: In this technique, the model is trained to differentiate between similar and dissimilar data points. Given two data samples, the model needs to predict whether they are from the same class or not.
- **Predictive Coding**: In this technique, the model is trained to predict future data points given a sequence of past data points. This is used for tasks like forecasting, where the goal is to predict future values of a time series, or video prediction where the goal is to predict the next frame of video given the previous frames.
- **Self-supervised pretraining**: this technique is used to train the model on some task with labels and then the trained model is used to initialize the weights of the model that is trained on the task of interest which is performed without any labels.

Along with several methods shared with the unsupervised strategy outlined in the previous subsection, such as:

- Clustering;
- Autoencoders;
- Generative Models.

These are some of the most common self-supervised learning techniques, but new methods and variations continue to be developed. Depending on the specific problem and dataset, one technique may be more suitable than the others.

Self-supervised learning and unsupervised learning are similar in that they both involve training a model on a dataset without the use of explicit labels. However, there are some key differences between the two techniques:

1. **Task-Specificity**: In self-supervised learning, the model is trained to perform a specific task, such as predicting missing words in a sentence or identifying particle tracks in detector images, while unsupervised learning techniques focus on discovering structure in the data or identifying patterns in the data, such as clustering.
2. **Labels**: Self-supervised learning uses the structure of the input data as a form of supervision, while in unsupervised learning no labels are used, the model solely relies on the patterns in the data.

3. **Inputs**: Self-supervised learning requires a specific form of inputs, usually it has the same input and output, such as a video, a sequence of words, or an image, but the task is to predict certain aspects of it. Unsupervised learning can be applied to any kind of input and the model is not trained to predict or reconstruct specific aspects of it.
4. **Pretraining**: self-supervised learning can be used as a pretraining step for other supervised tasks, where the model is fine-tuned with supervised data after self-supervised pretraining. Unsupervised learning is not used as a pretraining step because it doesn't perform any specific task.
5. **Evaluation**: In self-supervised learning, the model is evaluated based on how well it performs the specific task it was trained on, while in unsupervised learning, evaluation is more subjective and may depend on the specific research question or application.

In summary, self-supervised learning is a form of unsupervised learning that involves training a model to perform a specific task using the structure of the input data as a form of supervision, while unsupervised learning is focused on discovering structure or patterns in the data.

2.2 Anomaly Detection

Anomaly Detection (AD) [62], also known as outlier detection, is the task of identifying data points that deviate significantly from the rest of the data. These data points often referred to as anomalies, can be caused by errors in data collection, measurement or labelling, or can indicate rare or interesting events. Anomaly detection is a fundamental problem in many fields, including finance, healthcare, and cybersecurity, as well as HEP.

2.2.1 General Approach

Supervised, unsupervised and self-supervised learning approaches can be used for anomaly detection, depending on the availability of labelled data and the specific characteristics of the problem.

Supervised learning approaches rely on labelled training data to learn the patterns of normal events, and to identify events that deviate significantly from the normal patterns. These approaches are suitable when labelled data is available, but they may be sensitive to variations in the training data, and they may not be able to detect new types of anomalies.

Unsupervised learning approaches, on the other hand, do not rely on labelled data, and they aim to identify patterns in the data that are different from what is expected. These approaches are suitable when labelled data is not available, and they can detect new types of anomalies.

Self-supervised learning [63] can be used for anomaly detection as well. One common approach for self-supervised anomaly detection is to use *autoencoders*. The idea is to train an autoencoder on a dataset, and then use it to reconstruct new data points. Data points that are different from the norm will have a higher reconstruction error, indicating that they are anomalous. Another approach is to use generative models such as GANs or VAEs and use them to generate new data points. The generated data points are compared

to the real data points, and any data point that is dissimilar to the generated samples is considered anomalous. Another approach is to use contrastive learning [64], where the model is trained to differentiate between similar and dissimilar data points, once the model is trained the samples that the model is not able to tell that they are similar to other samples in the training set are considered anomalies. In predictive coding, the model is trained to predict future data points given a sequence of past data points, then in the test time if the model fails to predict the next data point or the prediction is far from the true next data point, it indicates an anomaly. In self-supervised anomaly detection, the model must learn to identify the normal behaviour from the data without the explicit use of labels. A related approach is semi-supervised anomaly detection, where only a subset of the data has labels. This scenario is practical when the background is contaminated by a small percentage of signal (e.g. $< 0.1\%$), as shown in the reference [65] where QCD jets are analyzed in the quest for anomalies in jet-substructure.

2.2.2 Various Techniques

There are different approaches to anomaly detection, but some common ones include:

- **Statistical-based** methods: These methods are based on the assumption that the normal data follows a certain probability distribution. The most common distribution used is the Gaussian distribution. The basic idea is to model the normal data as a Gaussian distribution and then use probability theory to identify any data points that have low probability according to the estimated distribution.
- **Distance-based** methods: These methods are based on the assumption that normal data points are closely grouped together and that outliers are farther away. The basic idea is to define a distance metric between data points and then identify any data points that are far away from the others. One of the most popular distance based methods is the Local Outlier Factor (LOF) which uses the local density of the data points to identify the outliers.
- **Clustering-based** methods: These methods are based on the assumption that normal data points are densely grouped, while outliers are not. The basic idea is to perform clustering on the data and then identify any data points that do not belong to any cluster.
- **Reconstruction-based** methods: These methods are based on the assumption that normal data can be reconstructed or reconstructed with low error while anomalies cannot. The basic idea is to reconstruct the normal data using an autoencoder, and then identify any data points that have high reconstruction errors. Variational Autoencoder is one of the most popular reconstruction based methods.
- **Neural Networks-based** methods: These methods are based on the assumption that the normal data and anomalous data have different representations. Anomaly detection can be cast as a one-class classification problem, where the goal is to learn a boundary that separates the normal data from the anomalous data. Autoencoder based architectures are also used for anomaly detection as well as GANs. One example of an anomaly detection method that uses GANs is *AnoGAN* [66]. It is an unsupervised method that can be used to detect anomalies in data sets. The basic idea behind AnoGAN is to train a GAN on a given data set, and then use the generator to create new samples that are similar to the training data. Anomalies

in the data can then be detected by measuring the similarity between the generated samples and the original data. AnoGAN has been used in various applications such as image and video anomaly detection, but it has also been used in other areas such as network intrusion detection and fault detection in industrial systems.

It's important to note that in anomaly detection, the anomalies to be detected are typically rare events, and the dataset is highly imbalanced. Because of this, the usual accuracy metric can be misleading. Other evaluation metrics such as precision, recall, F1-score, Receiver Operating Characteristic (ROC) curve and Area Under the Curve (AUC) are more informative in these scenarios.

2.2.3 Some applications in HEP

In High Energy Physics, anomaly detection is important in identifying rare events, such as new particles or new physics phenomena. These events are often characterized by certain features that distinguish them from the more common, background events.

In the context of **supervised learning**, the typical usage is in the search of events often characterized by certain features that distinguish them from the more common, background events. In this scenario, one approach would be to train a supervised learning model on a labelled dataset of known signal and background events. The model would learn to distinguish between the two types of events based on the input features, such as the energy, momentum, and spatial distribution of the particles. Once the model is trained, it can be applied to new, unlabeled data to identify events that are likely to be signal events, or that deviate significantly from the background distribution. For example, the Higgs Boson is hard to detect due to its extremely short lifetime and the fact that it decays into other particles which are difficult to detect as well. The challenge is to separate the Higgs Boson signal from the vast amount of background events that mimic the Higgs boson's decay. In this case, supervised learning can be used to train a classifier on simulated signal and background events, which are used to mimic real events. Another example is the search for dark matter, the hypothetical invisible matter that makes up the bulk of the universe, which weakly interacts with the normal matter, thus it is difficult to detect. One way to search for dark matter is to look for interactions between dark matter particles and normal matter particles in detectors. In this case, supervised learning can be used to train a classifier to differentiate between normal interactions and weakly interacting ones.

In the context of **unsupervised learning**, anomaly detection can be used to identify events that deviate significantly from the norm, such as rare events or background contamination. One approach would be to use clustering or dimensionality reduction to identify regions of the feature space that contain anomalous events. Another example is in the task of particle identification, where the goal is to classify particles based on their physical properties. Instead of using labelled training data, unsupervised learning methods can be used to identify patterns in the input data and learn the mapping from inputs to outputs. Another example is in tasks such as detecting rare events in collider experiments, where it is important to identify patterns in high-dimensional data that deviate from the expected background. The technique AnoGAN and similar methods are effective at identifying rare events in HEP experiments and have the potential to improve the sensitivity of current detection methods.

In the context of **self-supervised learning**, anomaly detection can be used for jet

tagging, identifying the substructure of jets, which are collimated sprays of particles produced in high-energy collisions, is an important task. A self-supervised learning approach to jet tagging is to use an autoencoder to learn a compact representation of the jet substructure, and then use the reconstruction error to identify jets that are dissimilar to the norm. Track segment finding: Self-supervised learning techniques can be used to identify track segments in detector images. One approach is to use autoencoders to learn a compact representation of the image, and then use reconstruction error as a measure of anomalousness, also contrastive learning can be applied to this task. Calorimeter energy regression: Calorimeter images can be used as inputs for a self-supervised deep learning model for energy regression. The model is trained to predict the energy deposit of a particle by analyzing patterns in the calorimeter images, without explicit labels indicating the energy of each image. Time-series Anomaly detection: In detector systems, the readouts are typically time-series, self-supervised learning can be used to predict future readouts and comparing the prediction with the true readouts can flag anomalies. Event classification: Event classification in HEP requires the identification of specific patterns in the data, and self-supervised learning can be used to identify those patterns and classify events.

2.3 Feed-Forward Deep Neural Networks (DNNs)

An overview of the operation of feedforward neural networks (FFNNs), or multilayer perceptrons (MLPs), a key component of contemporary deep learning [67], is given in this section. An introduction to the architecture is provided, highlighting the important elements. The training process is then covered, including, as was previously covered in section 2.1.4, well-known optimization methods such as gradient descent and its variations as well as the Backpropagation algorithm. A description of various activation functions and their impact on network performance will be presented as well.

2.3.1 Neural Networks Basics: Neurons and Architecture

An FFNN is a type of artificial neural network in which the information flows in a single direction, from input to output, without any feedback connections. They consist of an input layer, one or more hidden layers, and an output layer. Each layer of an FFNN is made up of a set of *neurons*, which are simply mathematical functions that take in a set of inputs, perform a computation, and produce an output. A neural network is composed of many neurons i stacked into layers, with the output of one layer serving as the input for the next. The function of each neuron $a_i(\boldsymbol{x})$, that takes as input $\boldsymbol{x}$, which is a vector of d input features $\boldsymbol{x} = (x_1, x_2, ..., x_d)$, varies depending on the type of non-linearity, i.e. **activation function**, used in the network. However, in most cases, $a_i(\boldsymbol{x})$ can be broken down into a linear operation that weighs the relative importance of the inputs and a non-linear transformation $\sigma_i(z)$. The linear operation typically takes the form of a dot product with neuron-specific weights $\boldsymbol{w}^{(i)} = (w_1^{(i)}, w_2^{(i)}, ..., w_d^{(i)})$ and re-centring with a neuron-specific bias $b^{(i)}$. This process can be represented mathematically as:

$$a_i(\mathbf{x}) = \sigma(z^{(i)}) = \sigma(w^{(i)} \cdot x + b^{(i)}) = \sigma(\mathbf{x^T} \cdot \mathbf{w^{(i)}}), \tag{2.7}$$

where $\mathbf{x} = (1, \boldsymbol{x})$ and $\mathbf{w}^{(i)} = (b^{(i)}, \boldsymbol{w}^{(i)})$, see Figure 2.5.

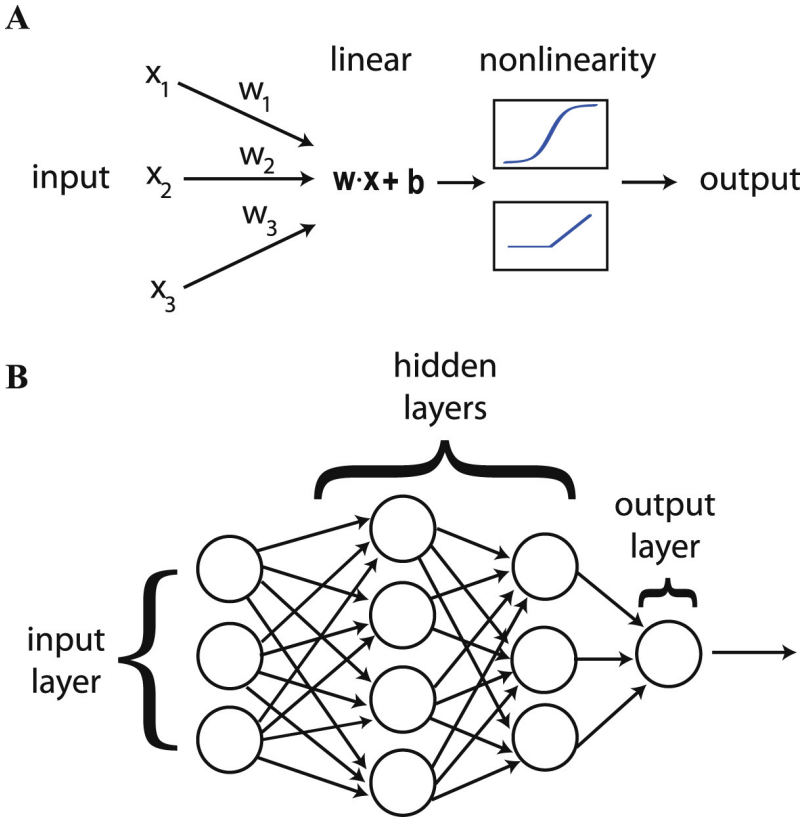


Figure 2.5: (A) The fundamental structure of neural networks includes stylized neurons, which are made up of a linear transformation that assigns a weight to various inputs, followed by a non-linear activation function. (B) These neurons are organized in layers, where the output from one layer serves as the input for the next layer [37].

The goal of a feedforward network is to approximate some function f^* [67]. An FFNN defines a mapping $y = f(\boldsymbol{x}; \theta)$ and learns the value of the parameters θ that result in the best function approximation. Feedforward neural networks are named as such due to their structure of connecting multiple functions. The model is represented by a directed acyclic graph which describes the composition of these functions. For instance, three functions $f^{(1)}$, $f^{(2)}$ and $f^{(3)}$ are connected in sequence, creating a chain $f(\boldsymbol{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x})))$. This chain-like structure is the most typical way neural networks are built. In this case, $f^{(1)}$ is the first layer, $f^{(2)}$ is the second layer and so on. The term *deep learning* comes from the concept of the **depth** of a neural network, which is determined by the number of layers in the model. In a feedforward network, the final layer is referred to as the **output layer**. The goal during training is to adjust the network's function, represented as $f(\boldsymbol{x})$, to match a desired function $f^*(\boldsymbol{x})$. The training data provides approximate examples of $f^*(\boldsymbol{x})$ at different points, each accompanied by a label $y \approx f^*(\boldsymbol{x})$. The training examples dictate the behaviour of the output layer at each point $\boldsymbol{x}$, but do not specify the behaviour of the other layers. It is up to the learning algorithm to decide how to use these layers to produce the desired output and approximate f^* because the training data does not provide output for these layers, they are referred to as **hidden layers**.

Neural networks are so named because they take inspiration from the structure and function of the human brain. Each hidden layer in the network is typically vector-valued, and the number of elements in the vector determines the **width** of the model. These elements can be thought of as playing a role similar to that of a neuron. The layers are

made up of many **units** that work in parallel and each unit represents a vector-to-scalar function and receives input from other units to compute its activation value. The concept of using multiple layers of vector-valued representation is inspired by neuroscience, and the functions used to compute these representations are also loosely based on observations of the functions of biological neurons. However, neural network research is guided by a variety of mathematical and engineering disciplines, and the goal is not to replicate the brain perfectly but to use function approximation to achieve statistical generalization and occasionally draw insights from our understanding of the brain.

The *activation* functions are a crucial component of neural networks. They are used to introduce non-linearity in the network, allowing the model to learn complex and non-linear relationships in the data. Common activation functions, shown in Figure 2.6 used are sigmoid, ReLU, LeakyReLU, softmax, tanh and ELU.

The **sigmoid function** is defined as:

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{2.8}$$

It is a smooth function that maps any input value to a value between 0 and 1. It is often used in the output layer of a binary classification problem. However, the sigmoid function has the drawback that it can saturate and cause the gradients to vanish, making it hard for the network to learn.

The rectified linear unit (**ReLU**) [68] is defined as:

$$f(x) = \max(0, x). \tag{2.9}$$

It is a simple and computationally efficient function that maps negative values to zero, and positive values to themselves. This function is less computationally expensive than the sigmoid function and does not saturate.

The **Leaky ReLU** [69] is defined as:

$$f(x) = \begin{cases} \alpha x & if x < 0 \\ x & otherwise \end{cases} \tag{2.10}$$

Where α is a small positive value, often set to 0.01. It is similar to the ReLU function, but it avoids the "dying ReLU" problem by allowing a small positive gradient when $x < 0$.

The Exponential Linear Unit (**ELU**) [70] activation function is an alternative to the ReLU activation function, which addresses the issue of *dying ReLU* by having a non-zero gradient for negative input values. The ELU activation function is defined mathematically as:

$$f(x) = \begin{cases} x & if x > 0 \\ \alpha(e^x - 1) & if x \leq 0 \end{cases} \tag{2.11}$$

where α is a hyperparameter, typically set to 1, that controls the steepness of the negative slope. The ELU activation function has been shown to improve the performance

of neural networks in some cases, particularly in image classification tasks. However, it is important to note that the ELU activation function is not always the best choice of activation function, and its use depends on the specific problem and dataset.

The **softmax** function is defined as:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}. \tag{2.12}$$

It is a generalization of the sigmoid function to multiple classes. It maps the input values to a probability distribution over the classes, with the property that the sum of the probabilities is 1. It is often used in the output layer of a multi-class classification problem. In statistical mechanics, the softmax function is related to the Boltzmann distribution, which describes the probability of a system being in a particular state based on its energy. The relationship between the softmax function and the Boltzmann distribution is that both functions squash a set of values down to a probability distribution. The softmax function can be viewed as a generalization of the Boltzmann distribution to multiple dimensions.

The **tanh** function is defined as:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.13}$$

It is similar to the sigmoid function but maps the input values to a range between -1 and 1. It is often used in the hidden layers of a neural network. Each of these activation functions has its advantages and disadvantages, and the choice of which one to use depends on the specific problem and the architecture of the network.

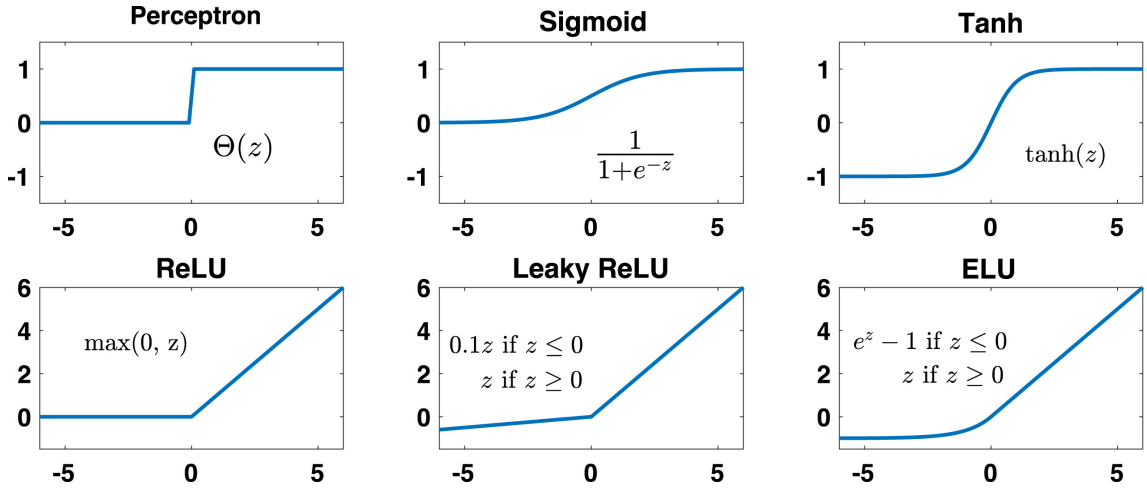


Figure 2.6: Non-linear activation functions, such as those in the bottom row, are commonly used in modern DNNs instead of saturating functions (top row) because they do not saturate for large inputs [37].

The *architecture* of an FFNN is defined by the number of layers, the number of neurons in each layer, and the connections between the neurons. The number of layers and neurons can be adjusted to suit the complexity of the problem at hand. The most common way to train an FFNN is through the *backpropagation* algorithm.

There are multiple variations and architectures such as convolutional neural networks (next section) and recurrent neural networks (RNNs) that build upon the basic architecture of an FFNN to address specific types of problems. One important thing to note is that, with the increase in data size and complexity of the problem, the training of FFNNs can be computationally expensive and time-consuming. That is the reason why, the use of more advanced optimization techniques like Adam, Adagrad, and Rmsprop (see section 2.1.4), which adapt the learning rate during training, are commonly used to speed up the process. In addition, techniques like dropout, weight decay, and batch normalization are also commonly used to prevent overfitting and improve the generalization of the model. The FFNNs are widely used in various domains, such as computer vision, natural language processing, speech recognition, and many others. They have been successful in tasks such as image classification, speech recognition, language translation and so on. However, FFNNs also have their limitations, such as the difficulty in modelling temporal dependencies and the inability to handle missing or inconsistent data. For this reason, other neural networks architectures like RNNs and LSTMs are more suitable for sequential data and handling missing data.

2.3.2 The Back-propagation Algorithm

The **back-propagation algorithm** [67, 71], often called **backprop**, is a widely used method for training neural networks. It is used to calculate the gradient of a continuous function, specifically the gradient of the cost function for the network's parameters. This gradient is then used by optimization algorithms such as stochastic gradient descent to adjust the parameters of the network to minimize the cost. The backpropagation algorithm is based on the idea of propagating information through the network in two directions:

- **forward** propagation step, input information flows through the network and produces an output;
- **backward** propagation step, the gradient of the cost function concerning the network's parameters is calculated by flowing the error information backwards through the network.

The backpropagation algorithm is particularly useful in deep learning, where neural networks have many layers. It is an efficient method for computing the gradient of the cost function for the parameters of the network, as it avoids the need for explicit calculation of the gradient through analytical methods, which can be computationally expensive. It is important to note that backpropagation is not a learning algorithm on its own, but rather a method for calculating gradients that are used in conjunction with optimization algorithms such as gradient descent. Additionally, it is not restricted to multilayer neural networks and can be applied to compute derivatives for any function. It is a very general technique that can also be used to compute other derivatives, such as the Jacobian of a function with multiple outputs.

Backpropagation uses the chain rule of calculus to compute the derivatives of composite functions, where the derivatives of the individual functions are known. It is based on the chain rule, which states that the derivative of a composite function can be calculated by multiplying the derivative of the outer function by the derivative of the inner function. Let x be a real number, and let f and g both be functions mapping from a real number to a real number. Suppose that $y = g(x)$ and $z = f(g(x)) = f(y)$. Then the chain rule states

that

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}. \tag{2.14}$$

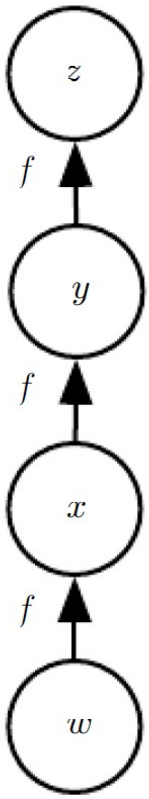


Figure 2.7: A computational graph that results in repeated subexpressions when computing the gradient [67].

To clarify the above definition of the back-propagation computation, let us consider the specific graph associated with a fully-connected multi-layer MLP. First shows the forward propagation, which maps parameters to the supervised loss $\mathcal{L}(\hat{y}, y)$ associated with a single (input, target) training example (x, y), with $\hat{y}$ the output of the neural network when x is provided in the input. Let $w \in \mathbb{R}$ be the input to the graph. We use the same function $f : \mathbb{R} \to \mathbb{R}$ as the operation that we apply at every step of a chain: $x = f(w)$, $y = f(x)$, $z = f(y)$. To compute $\partial z/\partial w$, we apply equation 2.14 and obtain:

$$\begin{aligned} \frac{\partial z}{\partial w} &= \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w} \\ &= f'(y)f'(x)f'(w) \\ &= f'(f(fw)))f'(f(w))f'(w) \end{aligned} \tag{2.15}$$

It is important to note that, the backpropagation algorithm can be affected by local minima and it's computationally expensive. That's why, variants of the backpropagation algorithm, such as the conjugate gradient method, the Levenberg-Marquardt algorithm, and the quick prop algorithm have been developed to improve the performance and speed

of the training process. Additionally, more advanced optimization algorithms like Adam, Adagrad and Rmsprop, which adapt the learning rate during training, can also be used to speed up the process.

It's also important to note that, the backpropagation algorithm requires a large amount of labelled data and it can be sensitive to noisy data. Therefore, techniques like regularization, early stopping, and dropout are commonly used to prevent overfitting and improve the generalization of the model.

2.3.3 Deep Double Descend

The double descent phenomenon in deep learning refers to the observation that the generalization error of deep neural networks (DNNs) can exhibit a non-monotonic behaviour as a function of the model's capacity. This means that as the model's capacity increases, the generalization error can initially decrease, reach a minimum, and then increase again [72]. In the under-parameterized regime, where the model complexity is small compared to the number of samples, the test error as a function of model complexity follows the U-shaped behaviour predicted by the traditional bias-variance trade-off (see section 2.1.3). However, once the model's complexity is sufficient to interpolate the training data and achieve zero training error or close, increasing the model's complexity further only decreases the test error (Figure 2.8), following the idea that "bigger models are better". This behaviour is in contrast to the classical bias-variance trade-off where generalization error is expected to decrease monotonically as the model's capacity increases, hence conventional wisdom in classical statistics is that, once we pass a certain threshold, "larger models are worse". Double descent has been observed in a variety of settings, such as the over-parameterized regime, where the number of parameters in the model is much larger than the number of training examples, and the interpolation regime, where the model can fit the training data perfectly. In both cases, the model has more capacity than necessary to fit the training data, and the additional capacity can lead to improved generalization performance on unseen data.

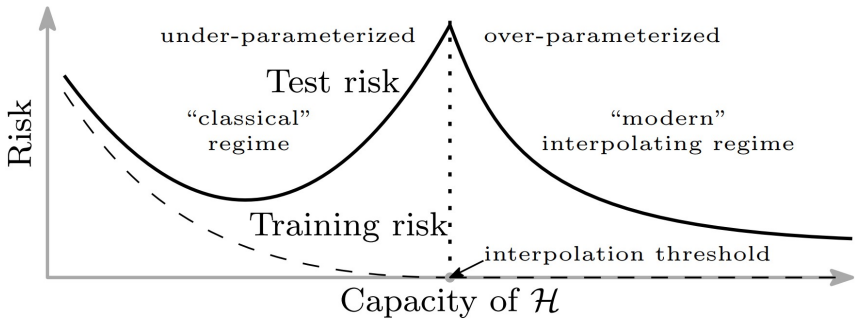


Figure 2.8: The double descent risk curve illustrates the relationship between the risk and the model capacity, it encompasses both the U-shaped risk curve that characterizes the "classical" regime (Figure 2.2) together with the behaviour observed with high capacity functions classes, the "modern" interpolating regime. The curve is separated by an interpolation threshold, the predictors on the right side of the threshold have zero training risk [72].

One possible explanation for double descent is the existence of a phase transition in the model's optimization landscape. In the over-parameterized regime, the optimization

landscape is relatively smooth and has many good solutions. As the model's capacity increases, the optimization landscape becomes more complex with more local minima. At a certain point, the optimization algorithm can get stuck in poor local minima, leading to increased generalization error. However, when the number of parameters surpasses a certain threshold, the optimization algorithm can escape poor local minima and find good solutions again, leading to improved generalization performance. Another explanation for double descent is that when the model capacity is increased, it is also able to learn more complex features, which are more robust to variations in the data, this robustness leads to better generalization.

It's important to notice that the double descent is not universal and there are cases where there is no such behaviour and a monotonic decrease of the generalization error is found. The double descent phenomenon is a relatively new and active area of research, and there is still much that is not understood about it. However, the general understanding is that double descent is a result of the complex and non-convex optimization landscape of DNNs, which is not well understood yet. It's an important topic in the field of deep learning, and it has implications for the design and training of deep neural networks, as well as the interpretation of generalization performance.

2.4 Convolutional Neural Networks (CNNs)

In this section, it is examined the structure of a Convolutional Neural Network (CNN). Some specifics of the convolutional layers will be shown, which are responsible for detecting features in the input data, the pooling layers, which are responsible for reducing the spatial dimensions of the data, and the fully connected layers, which make the final predictions. Investigation of the mathematical concepts and operations that define these layers, and how they work together to form a CNN.

2.4.1 Overview of the CNNs structure

The core principle of physics is that we should take advantage of symmetries and invariances when studying physical systems. Properties such as locality and translational invariance are often built into the physical laws themselves. Our models in statistical physics often incorporate all that we know about the physical system being studied. This idea of utilizing the additional structure in our analysis is a fundamental feature of modern physical theories, ranging from general relativity to gauge theories and critical phenomena. Similarly, many datasets and supervised learning tasks also possess symmetries and structures. For example, in a supervised learning task where the goal is to label images as pictures of cats or not, the statistical procedure must first learn features associated with cats, which are likely to be local, and translation invariance is built into the task. This example shows that, like physical systems, many machine learning tasks, particularly in computer vision and image processing, also possess additional structures such as locality and translational invariance.

A convolutional neural network, or CNN, is a neural network architecture that is designed to maintain translation invariance while also considering the spatial structure of the input data. CNNs are widely used in a variety of deep learning applications and are considered to be a fundamental component of modern deep learning. The building blocks of a CNN are composed of two types of layers: convolutional layers that perform

convolution operations on the input with a set of filters, and pooling layers that reduce the spatial resolution of the input while preserving important spatial information, as illustrated in Figure 2.9.

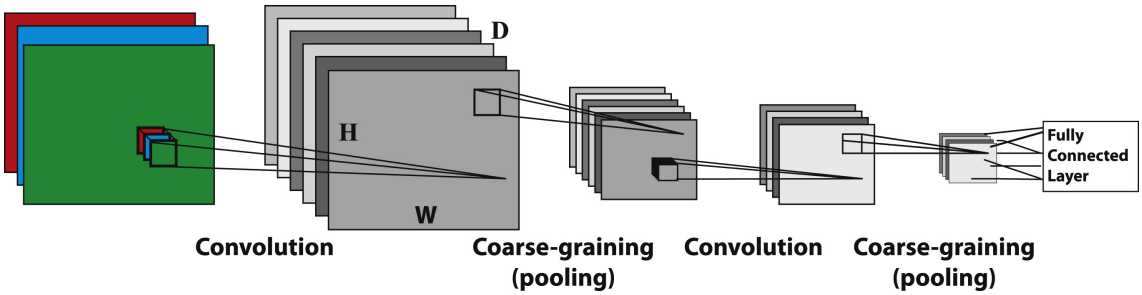


Figure 2.9: A Convolutional Neural Network (CNN) is structured in three dimensions: height (H), width (W), and depth (D). The input layer's depth corresponds to the number of channels (3 for RGB images). Neurons in the convolutional layers convolve the image with a local spatial filter at a given location in the $(W; H)$-plane. The depth D of the convolutional layer represents the number of filters used in that layer. Neurons at the same depth correspond to the same filter. The convolutional layer mixes inputs at different depths while maintaining the spatial location. Pooling layers perform a spatial coarse-graining at each depth, reducing the height and width while retaining the depth. The fully connected layer and classifier follow the convolutional and pooling layers (not shown) [37].

In detail, a convolutional layer in a CNN is characterized by its height, width, and depth. The height and width correspond to the size of the two-dimensional spatial plane in neurons, while the depth corresponds to the number of filters in that layer. These filters are applied locally over small spatial patches of the previous layer and are moved across the entire spatial plane with a specified stride. In this way, the convolution operation computes the dot product between the filter weights and the input values at each location. The output of this operation is a feature map that represents the response of the filter to the input at each location. A 2-dimensional layer is characterized by 3 numbers: the height (H_l), width (W_l), and depth (D_l) of the layer. The height and width correspond to the spatial dimensions of the layer (in neurons), and the depth D_l corresponds to the number of filters in that layer. All neurons associated with a particular filter have the same parameters (i.e. shared weights and bias). Typically, local spatial filters (referred to as receptive fields) are used, which take as input a small spatial patch from the previous layer at all depths. To illustrate how this works, consider the simple example of a 1-dimensional input of depth 1, shown in Fig. 43. In this case, a filter of size $F \times 1 \times 1$ can be represented by a vector of weights w with length F. The stride (S) indicates how many neurons the filter is translated by when performing the convolution.

Convolutional layers are typically interleaved with pooling layers that reduce the spatial resolution of the feature maps by performing a subsampling operation. One common pooling technique is max pooling, in which a small region of the feature map is replaced by a single neuron whose output is the maximum value within that region. This reduces the dimension of the output, for example by halving the height and width of the pooling region is 2×2. Pooling is typically performed separately at each depth, so it does not reduce the depth of the feature maps. An example of max pooling is shown in the figure. In a CNN, the convolution and max-pooling layers are typically followed by a fully connected layer

[73]. This layer connects all the neurons in the previous layer to the next layer, allowing for the use of a standard backpropagation algorithm for training. From a training perspective, CNNs are similar to fully connected neural network architectures except for the shared parameters in the convolutional layers.

The convolutional structure of CNNs not only introduces additional features such as translational invariance and locality but also has numerous practical and computational advantages. In a CNN, all neurons in a given layer use the same filter, which can be represented by a single set of weights and biases. This reduces the number of free parameters by a factor of $H \times W$ per layer, enabling the training of much larger models than with fully connected layers. This concept is similar to how in physics, translationally invariant systems can be represented by specifying only their momentum and functional form, whereas without translation invariance, more information is required.

To expand on the impact of CNNs, it's worth noting that their success in computer vision tasks has led to a wide range of applications in various fields. For example, they have been used for object detection and recognition, facial recognition, medical image analysis, autonomous driving, and more.

Moreover, CNNs have also inspired the development of other neural network architectures that have further advanced the field of deep learning. For instance, the concept of residual connections introduced in ResNet [74] has enabled the training of even deeper networks, while attention mechanisms [75] in Transformer models have led to breakthroughs in natural language processing tasks.

Overall, the importance of CNNs in computer vision cannot be overstated. They have not only demonstrated their effectiveness in a wide range of applications, but they have also influenced the development of other neural network architectures that have advanced the field of deep learning as a whole. Some examples are:

- **Object recognition**: CNNs have shown remarkable success in identifying objects within images or videos, even when they are partially occluded or in complex scenes.
- **Semantic segmentation**: This involves labeling each pixel in an image with its corresponding class, which is useful for tasks like autonomous driving or medical image analysis.
- **Depth estimation**: CNNs can estimate the depth or distance of objects within an image or video, which is useful for 3D reconstruction or augmented reality applications.
- **Image super-resolution**: CNNs can be trained to generate high-resolution images from low-resolution inputs, which is useful for applications like satellite imagery or medical imaging.
- **Human pose estimation**: CNNs can estimate the 2D or 3D positions of human body joints from an image or video, which is useful for applications like motion capture or sports analytics.

As well as applications in HEP, CNNs have been used for tasks such as particle classification and event reconstruction in particle physics experiments like the LHC. For example, the ATLAS and CMS experiments have used CNNs to identify Higgs boson events and improve the efficiency and accuracy of particle identification [76].

2.4.2 Normalization Methods

Normalization layers are commonly used in CNNs to improve the training and generalization performance of the model. In particular, normalization layers help to alleviate the *internal covariate shift problem*, which is the phenomenon where the distribution of feature activations changes during the training process. This shift in distribution could happen because the inputs to each layer depend on the activations of the previous layers, which change as the model is being trained. As a result, the activations of each layer may have a different distribution, which can cause the optimization process to slow down or even prevent the model from converging to a good solution and suboptimal results. Batch Normalization [77] and Instance Normalization [78] are two popular types of normalization layers used in CNNs for machine vision tasks.

Batch Normalization

Batch Normalization (BN) addresses this problem by normalizing the activations of each batch during training. Specifically, it computes the mean and variance of the activations within a batch, and then applies a transformation to center the activations around zero and scale them by the variance of the batch. This helps to ensure that the activations have a similar distribution across all batches and can help to reduce the internal covariate shift problem. The Batch Normalization transformation can be described by the formula:

$$\begin{aligned} \hat{x}_i &= \frac{x_i - \mathbb{E}[x]}{\sqrt{\mathbb{V}[x] + \epsilon}}, \\ y_i &= \gamma \hat{x}_i + \beta; \end{aligned} \tag{2.16}$$

where x is the input to the BN layer, $\mathbb{E}[x]$ and $\mathbb{V}[x]$ are the mean and variance of the batch, $\hat{x}_i$ is the normalized activation, γ and β are learnable scaling and shifting parameters, and ϵ is a small constant added for numerical stability. The first term of the formula, $\frac{x_i - \mathbb{E}[x]}{\sqrt{\mathbb{V}[x]+\epsilon}}$, is the normalization step. Here, x_i represents the i-th activation in the batch, $\mathbb{E}[x]$ represents the mean activation of the batch, and $\mathbb{V}[x]$ represents the variance of the batch. The term ϵ is added to the variance to prevent division by zero. This step normalizes each activation so that it has zero mean and unit variance, which helps to stabilize the optimization process. The second term of the formula, $\gamma \hat{x}_i + \beta$, is the scaling and shifting step. Here, γ and β are learnable parameters that allow the model to learn the optimal scale and shift for each normalized activation. This step helps to reintroduce some of the original distribution and variability of the activations, which can help to improve the expressive power of the model.

Instance Normalization

Instance Normalization (IN) is a normalization technique that is similar to BN but normalizes the activations of each instance (i.e., channel) in the input rather than the activations of each batch. This normalization is commonly used in style transfer and image-to-image translation tasks where the spatial correlations between feature maps are important. This technique has also been used for anomaly detection in computer vision because it can help to reduce the variability of the feature maps and make them more consistent across

different images. One way to use IN for anomaly detection is to apply it to the feature maps extracted by a pre-trained CNN, which can help to normalize the feature maps across channels. Then, a classifier or anomaly detection algorithm can be applied to the normalized feature maps to detect anomalies. For instance, if a pre-trained CNN is used to extract features from images, the feature maps can be normalized using IN before being passed to a classifier or anomaly detection algorithm. By applying IN, the feature maps are made more consistent across different images, which can help to improve the accuracy and robustness of the anomaly detection algorithm.

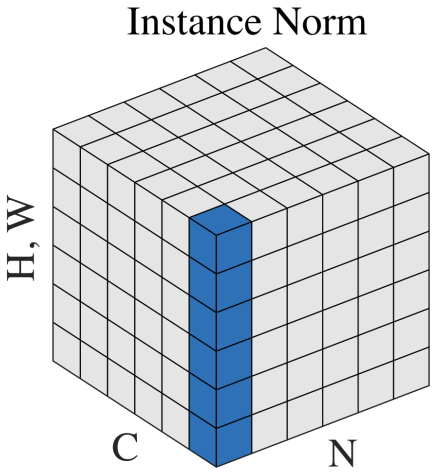


Figure 2.10: Instance Normalization method. This plot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels [79].

The main difference between BN and IN is that BN normalizes each feature map across the entire batch, while IN normalizes each feature map across spatial locations, see figure 2.10. Therefore, IN is more effective in preserving the style and texture of the input image. It is particularly useful when the input image has a distinct style or texture that needs to be preserved or transferred to another image. The formula for instance normalization is similar to that of batch normalization, but it operates on each instance or channel independently. For a 3D input tensor x, the instance normalization formula can be written as:

$$\begin{aligned} \hat{x}_{i,j,k} &= \frac{x_{i,j,k} - \mathbb{E}[x_{i,\cdot,\cdot}]}{\sqrt{\mathbb{V}[x_{i,\cdot,\cdot}] + \epsilon}}, \\ y_{i,j,k} &= \gamma_i \hat{x}_{i,j,k} + \beta_i. \end{aligned} \tag{2.17}$$

Here, $x_{i,j,k}$ is the activation of the k-th channel at spatial location (i, j) in the input, $\mathbb{E}[x_{i,\cdot,\cdot}]$ and $\mathbb{V}[x_{i,\cdot,\cdot}]$ are the mean and variance of the i-th channel, $\hat{x}_{i,j,k}$ is the normalized activation, γ_i and β_i are learnable scaling and shifting parameters for the i-th channel, and ϵ is a small constant added for numerical stability.

Overall, normalization layers such as BN and IN can help to improve the training and generalization performance of CNNs for machine vision tasks by reducing the internal

covariate shift problem. BN is used to normalize the activations of each batch during training, while IN is used to normalize the activations of each instance (i.e., channel) in the input. These normalization techniques have proven to be effective in improving the convergence rate, accuracy, and robustness of CNNs.

2.4.3 Regularization Methods

This section of the thesis will discuss various regularization techniques commonly used in convolutional neural networks (CNNs). Regularization methods are employed to prevent overfitting such as weight decay, dropout, early stopping, data augmentation and ReduceLROnPlateau.

Weight Decay

Weight decay is a form of regularization that penalizes large weights in the model during training. It is also sometimes referred to as L2 regularization or ridge regression. The basic idea behind weight decay is to add a regularization term to the loss function that encourages the model to have small weight values. This is achieved by adding a penalty term proportional to the square of the magnitude of the weight values. The loss function with weight decay can be written as:

$$\mathcal{L}_{wd} = \mathcal{L}(\mathbf{x}, \mathbf{x}') + \frac{\lambda}{2}||\mathbf{w}||_2^2 \tag{2.18}$$

where $\mathcal{L}$ is the original loss function, $\mathbf{x}$ is the ground truth label, $\mathbf{x}'$ is the predicted label, $\mathbf{w}$ is the weight vector of the network, and λ is a hyperparameter that controls the strength of the weight decay penalty. The weight decay penalty term encourages the model to have small weight values, which in turn can prevent overfitting. This is because large weights can cause the model to become overly complex and sensitive to noise in the training data, while small weights are less likely to be influenced by noise.

One practical example of weight decay in CNNs is the use of the Adam optimizer with weight decay. The Adam optimizer is a popular optimization algorithm that uses both momentum and adaptive learning rates to speed up convergence. To add weight decay to the Adam optimizer from equation 2.3, we simply add a penalty term to the update rule for the weights and in turn we obtain the AdamW, i.e. the equation 2.6.

Dropout

Dropout is a regularization technique that addresses the problem of overfitting in neural networks. The basic idea behind dropout is to randomly drop out, i.e. set to zero, a proportion of the neurons in a layer during training. By doing so, the network is forced to learn more robust and generalizable features, as it cannot rely on any one particular neuron to always be present. The dropout technique is applied during training by randomly selecting a subset of neurons to be dropped out in each forward pass. The probability of dropping out a neuron is a hyperparameter that can be tuned based on the specific task and network architecture. Dropout can be applied to both input and hidden layers, although it is typically more effective in hidden layers.

A more mathematical description can be expressed considering h be the input to a dropout layer, and p be the probability of dropping out a neuron. During training, each neuron i in h is set to zero with probability p, independently of other neurons. The output y of the dropout layer is then scaled by $1/(1-p)$ to ensure that the expected value of y remains the same during training and testing.

$$y_i = \begin{cases} 0 & \text{with probability } p \\ \frac{h_i}{1-p} & \text{otherwise} \end{cases} \tag{2.19}$$

During testing, all neurons are used, but their activations are scaled by $(1-p)$ to account for the fact that more neurons are active than during training. This ensures that the expected output during testing remains the same as during training. Dropout has been shown to be an effective regularization technique in a variety of DL tasks, including image classification and speech recognition. However, it can increase the training time of the network, as more iterations are needed to converge to the optimal solution. Nonetheless, the benefits of dropout often outweigh the computational cost, making it a popular choice for regularization.

Early Stopping

Early stopping is a regularization technique that involves monitoring the performance of a model during training and stopping the training process when the performance on a validation set stops improving. The basic idea is that, as the model trains, it begins to overfit the training data and memorize noise in the data. This causes the performance on the validation set to start decreasing, even as the performance on the training set continues to improve. By monitoring the validation set performance and stopping training when it stops improving, we can prevent the model from overfitting and improve its generalization performance. We can express the early stopping criterion as follows:

$$\hat{\theta} = \arg\min_{\theta \in \Theta} \mathcal{L}(\theta; X_{train}, X_{train}) + \lambda \mathcal{V}(\theta; X_{val}, X_{val}) \tag{2.20}$$

where Θ is the space of model parameters, $\mathcal{L}$ is the loss function on the training set, $\mathcal{V}$ is the validation loss function, and λ is a regularization parameter that controls the trade-off between the training loss and the validation loss. During training, we monitor the validation set performance after each epoch and stop training when the validation loss has not improved for a certain number of epochs. The value of λ can be determined using cross-validation. Early stopping is a simple and effective regularization technique that has been shown to improve the generalization performance of deep learning models. It is particularly useful when training large and complex models on limited training data.

Data Augmentation

Data augmentation is a regularization technique that artificially increases the size of the training dataset by generating new training examples from the existing ones. This technique is particularly useful when the size of the training dataset is limited, as it helps prevent overfitting by providing the network with more diverse examples to learn from. In the context of CNNs, data augmentation can be implemented in many ways. Common methods include rotation, translation, flipping, and scaling of the input images. For example, an

image of a cat can be rotated by a certain angle to generate a new image, which can be used to train the CNN. Similarly, an image can be flipped horizontally or vertically, or cropped to create a new training example. Data augmentation can be represented by a function $T(x)$ that transforms the input image x into a new image x', such that $T(x)$ is a stochastic function that produces different outputs for the same input. The transformed image x' is then used as a training example for CNN.

$$x' = T(x) \tag{2.21}$$

The choice of the transformation functions T depends on the specific application and the characteristics of the input data. For example, for image classification tasks, typical transformations include random rotations, translations, and flips. Data augmentation is a powerful technique for improving the performance of CNNs, especially when the size of the training dataset is limited. By providing the network with more diverse examples to learn from, data augmentation can help prevent overfitting and improve the generalization performance of the model.

ReduceLROnPlateau

ReduceLROnPlateau is a regularization technique used in training deep learning models, including CNNs, to improve their performance. This technique is based on dynamically reducing the learning rate of the optimizer when a metric, such as validation loss or accuracy, stops improving. The idea is to prevent the model from getting stuck in a suboptimal solution by reducing the learning rate and allowing it to explore other areas of the optimization landscape.

The ReduceLROnPlateau technique is implemented using a callback function in the training process, which monitors the selected metric and reduces the learning rate when the metric stops improving. The new learning rate η_{new} is calculated as a fraction of the previous learning rate η_{old}:

$$\eta_{new} = \eta_{old} \times \texttt{factor} \tag{2.22}$$

where the `factor` is typically set to a value less than one, such as 0.1 or 0.5, to reduce the learning rate by a significant amount. The ReduceLROnPlateau technique is particularly useful when training deep CNNs on large datasets, where the optimization landscape can be complex and non-convex. By reducing the learning rate, the optimizer can take smaller steps towards the optimal solution and potentially avoid getting stuck in local minima.

2.5 Autoencoders (AEs)

Autoencoders are a type of neural network architecture that is used for efficient encoding in unsupervised learning context [80]. The goal of training an autoencoder is to learn a compact representation of the input data that can be used for tasks such as dimensionality reduction, denoising, or generative modelling.

2.5.1 AEs Architecture and Operation

Autoencoders [67] are neural networks that are trained to reconstruct their input. The basic architecture, shown in Figure 2.11, consists of an *encoder* network, which maps the input data to a lower-dimensional representation, typically made via the imposition of a *bottleneck* in the network structure and a *decoder*, which maps the lower-dimensional representation back to the original input space. Compression and reconstruction are extremely demanding if there is no structure in the data, i.e. no correlation between input features. However, if some sort of structure exists in the data, it can be learned and applied when forcing the input through the bottleneck. Encoder and decoder functions are:

- **data-specific**, so they will only be able to compress data that is similar to what they trained on;
- **lossy**, which means that the decompressed outputs will be degraded compared to the original inputs;
- **learned automatically from data examples**, it means that it is easy to train specialised instances of the algorithm, that will perform well on a specific type of input.

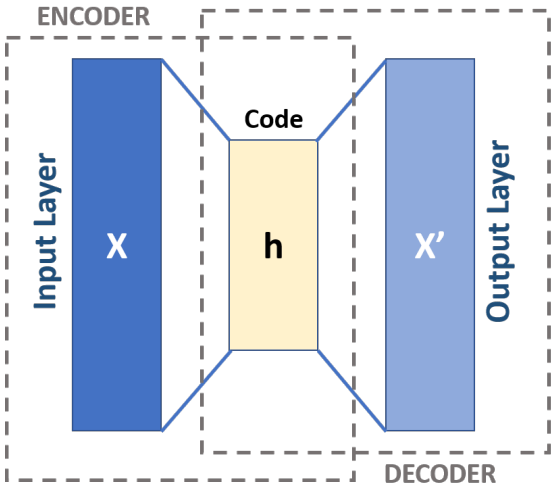


Figure 2.11: Diagram of a basic Autoencoder.

Typically, they are restricted in a way that allows them to copy only approximately and to copy only input that resembles the training data. By prioritizing which parts of the input to copy, the model learns useful properties of the data. Traditionally, autoencoders were used to reduce dimensionality or to learn features. Recently, theoretical connections between autoencoders and latent variable models have brought autoencoders to the forefront of generative modelling.

A mathematical description of an AE can be done as follow: considering the input x, passed through the encoder function f, which maps it to a lower-dimensional feature space $z = f(x)$. The feature space is then passed through the decoder function g, which maps it back to the original input space $x' = g(z)$. The output of the decoder x' is compared to the original input x, using a *reconstruction loss* function $\mathcal{L}$, as it is a check of how well the input x has been reconstructed from the input. The Learning process is described simply as minimizing a loss function:

$$\mathcal{L}(x, g(f(x))), \tag{2.23}$$

where the loss function $\mathcal{L}$ penalizes $g(f(x))$ for being dissimilar from x, such as the mean squared error. It is worth noting that autoencoder architecture can be varied to improve performance, such as by adding a bottleneck layer or using a denoising autoencoder.

It is important to note that the above explanation is a basic one, and there are many variations and extensions of autoencoders, such as denoising autoencoder, variational autoencoder, and others.

2.5.2 Variations

There are several types of autoencoders based on the dimension of the latent space:

- **Undercomplete autoencoders** have a bottleneck structure where the dimension of the latent space is less than the dimension of the input data. This type of autoencoder is trained to compress the data by learning a lower-dimensional representation of the input.
- **Regular autoencoders** have a latent space of the same dimension as the input data. This type of autoencoder is trained to reconstruct the input data by learning a representation that is a function of the input.
- **Overcomplete autoencoders** have a latent space that is larger than the dimension of the input data. This type of autoencoder is trained to learn a higher-dimensional representation of the input.
- **Convolutional Autoencoder (ConvAE)** uses convolutional layers instead of fully connected layers to learn a compact representation of the input data. The main advantage of using convolutional layers is that they can take advantage of the spatially local correlations in the data, which is particularly useful for images and other grid-like data. In addition to using convolutional layers, a ConvAE also has a convolutional bottleneck, meaning that the latent space or codings are a three-dimensional tensor (or four-dimensional if you consider the batch dimension) with the same spatial dimensions as the input data, i.e. it is of the form $H \times W \times D$. The encoder is typically composed of a series of convolutional layers followed by pooling layers, which reduce the spatial dimensions of the data while increasing the depth. The decoder is typically composed of a series of transposed convolutional layers (also known as upsampling layers) that increase the spatial dimensions of the data while decreasing the depth.
- **Sparse autoencoder (SAE)** which is trained to produce sparse representations of its input data. The sparse constraint is typically imposed by adding a sparsity regularization term to the loss function that the autoencoder is being trained to minimize. The most common form of sparsity regularization is the L1 regularization, also known as the Lasso, which encourages the model to produce sparse representations by adding a term to the loss function that is proportional to the sum of the absolute values of the activations of the hidden units. The L1 regularization term can be written as $\lambda \sum_{i=1}^{n} |a_i|$, where a_i is the activation of the i-th hidden unit and $\lambda > 0$ is a hyperparameter that controls the strength of the regularization measures how much sparsity we want to enforce.

- **Denoising Autoencoder (DAE)** is a type of autoencoder that is trained to reconstruct the original input data from a corrupted version of the input. This type of autoencoder is useful for data cleaning and denoising tasks. The problem of training a DAE is the optimization problem: $\min_\theta \mathcal{L}(\theta)$, where θ are the parameters of the autoencoder.
- **Contractive autoencoder (CAE)** which modifies the objective function to encourage the encoder to learn a more robust and compact representation by adding a penalty term on the L_2 norm of the Jacobian of the encoder concerning its inputs, expressed as $\lambda|J(g_\theta(x_i))|_F^2$, where λ is a hyperparameter, $J(g(x))$ is the Jacobian of the encoder function and $|.|_F$ is the Frobenius norm of a matrix.
- **Variational Autoencoders (VAE)** which are a type of variational Bayesian method that has a similar architecture to basic autoencoders but has distinct goals and a distinct mathematical formulation. Instead of having a fixed vector, the latent space in VAEs is composed of a mixture of distributions. This topic will be explored in more detail in the next section of this thesis.

All these variations of autoencoder have their advantages and disadvantages, and they are particularly useful in different scenarios.

2.6 Variational AutoEncoders (VAEs)

Variational Autoencoders (VAEs) are a popular generative model that combines the idea of autoencoders with probabilistic modelling. They learn a compact latent representation of the input data and can generate new samples from it.

2.6.1 VAEs Architecture and Operation

A standard Variational Autoencoder (VAE) [53] is an artificial neural network architecture that aims to learn a compact, continuous latent representation of data. The basic idea behind a VAE is to learn an *encoder* function $q_\phi(z|x)$, that maps the input data x to a latent representation z and a *decoder* function $p_\theta(x|z)$ that maps the latent representation back to the original data space, schematic is depicted in Figure 2.12.

The encoder is typically modelled as a neural network with parameters ϕ, that maps the input x to a mean μ and a standard deviation σ of a Gaussian distribution over the *latent space representation* z. The encoder can be formulated as:

$$q_\phi(z|x) = \mathcal{N}(z|\mu_\phi(x), \sigma_\phi(x)), \tag{2.24}$$

and the decoder function $p_\theta(x|z)$ is also modeled as a neural network with parameters θ, that maps the latent representation z back to the original data space. The decoder can be formulated as:

$$p_\theta(x|z) = p_\theta(x|f_\theta(z)), \tag{2.25}$$

where $f_\theta(z)$ is the output of the encoder neural network. The goal is to learn the encoder and decoder parameters such that the model can generate new samples from the learned latent space by sampling from the *prior* $p(z)$ and passing the samples through the

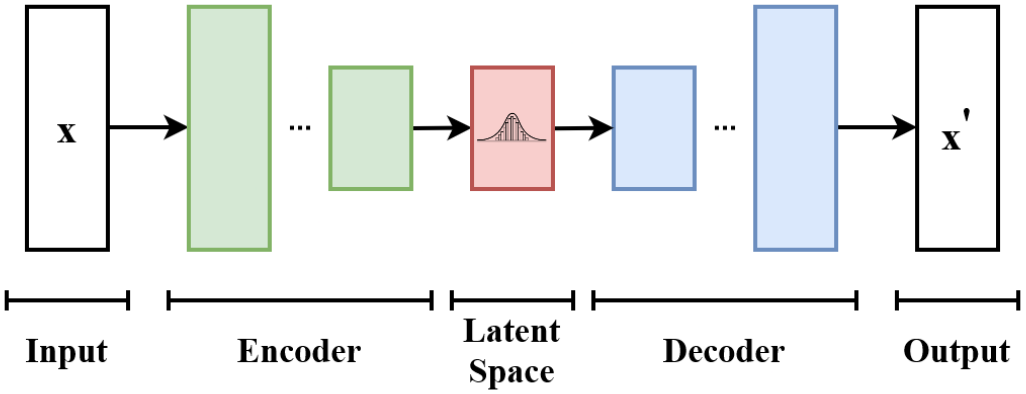


Figure 2.12: The variational autoencoder's basic design. x is provided to the model as input. It is compressed by the encoder and stored in latent space. The information collected from the latent space is fed into the decoder, which then outputs x' that is as close to x as possible.

decoder. The key difference between a VAE and a traditional autoencoder is the use of a probabilistic encoder and a prior over the latent space. A traditional autoencoder uses a deterministic encoder to map the input to a fixed-size latent representation, whereas a VAE uses a probabilistic encoder that maps the input to a probability distribution over the latent space.

In general, it is only possible to see an observation x, but we would like to infer the characteristics of z [81]. In other words, the relationship between the input data and its latent representation is given by the computation of $p_\theta(z|x)$, according to Bayes' theorem:

$$p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)}, \tag{2.26}$$

where $p_\theta(z)$ is the prior, $p_\theta(x|z)$ is the likelihood and posterior $p_\theta(z|x)$. From the point of view of probabilistic modelling, the goal is to maximize the likelihood of the data x by their chosen parametrized probability distribution $p_\theta = p(x|\theta)$. Let us find $p_\theta(x)$ via marginalizing over z.

$$p_\theta(x) = \int_z p_\theta(x, z)dz = \int_z p_\theta(x|z)p_\theta(z)dz, \tag{2.27}$$

where $p(x, z)$ represents the joint distribution under p_θ of input data x and its latent z. The second inequality is given by the chain rule.

Unfortunately, computing $p(x)$ is quite difficult. This usually turns out to be an intractable distribution[1]. However, we can apply variational inference [82] to estimate

[1] in this context *intractability* means one of the two things

- the integral has no closed-form solution. This might be when we are modelling some complex, real-world data and it is so complex that we cannot write down a likelihood (or if we can it takes forever to evaluate), but we can simulate from the data generating process (e.g. some kind of process

this value. Let's approximate $p(z|x)$ by another distribution $q(z|x)$ which we will define such that it has a tractable distribution. If it is defined the parameters such that $q_\phi(z|x) \approx p_\theta(z|x)$, with ϕ defined as the set of real values that parametrize q. This distribution can be used to perform approximate inference of the intractable distribution. In this way, the problem is of finding a good probabilistic autoencoder, in which the conditional likelihood distribution $p_\theta(x|z)$ is computed by the probabilistic decoder, and the approximated posterior distribution $q_\phi(z|x)$ is computed by the probabilistic encoder. To measure the difference between two probability distributions we use the Kullback-Leibler (KL) divergence, also called relative entropy or I-divergence [83], denoted as $D_{KL}(P \parallel Q)$. It is a type of statistical distance: a measure of how one probability distribution P is different from a second, reference probability Q of a continuous random variable, defined to be the integral:

$$D_{KL}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx, \tag{2.28}$$

where p and q denote the probability densities of P and Q. KL divergence measures the amount of information lost when approximating P with Q, always non-negative and also asymmetric, i.e. $D_{KL}(P \parallel Q) \neq D_{KL}(Q \parallel P)$. This distance is used in the variational autoencoder to measure the difference between the approximate posterior and the true posterior. In VAEs the KL divergence between the approximate posterior and a prior over the latent variables is added as a regularization term to the standard reconstruction loss, called the *evidence lower bound* (ELBO). The ELBO is maximized for the parameters of the encoder and the decoder network, which results in learning a generative model. The reparameterization trick is a technique to make the gradients of the KL divergence term in the ELBO estimable via backpropagation. It transforms the random sampling process into a differentiable transformation of the parameters of the approximate posterior.

2.6.2 Evidence Lower Bound (ELBO)

The goal is to learn the encoder and decoder parameters ϕ and θ such that the model can generate new samples from the learned latent space by sampling from the prior $p(z)$ and passing the samples through the decoder. To achieve this, the model is trained via a differentiable loss function to update the network weights through backpropagation, to minimize the reconstruction error between the original data x and the generated data x'.

Thus, to ensure that $q_\phi(z|x)$ is similar to $p_\theta(z|x)$, we could minimize the KL divergence between the two distributions:

for how certain properties develop over many generations in a population).

- The integral is computationally intractable, i.e. exponential distribution.

$$
\begin{aligned}
D_{KL}(q_\phi(z|x) \parallel p_\theta(z|x)) &= \mathbb{E}_{z\sim q}\left[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\right] \\
&= \mathbb{E}_{z\sim q}\left[\log \frac{q_\phi(z|x)p_\theta(x)}{p_\theta(x,z)}\right] \\
&= \log p_\theta(x) + \mathbb{E}_{z\sim q}\left[\frac{q_\phi(z|x)}{p_\theta(x,z)}\right].
\end{aligned} \tag{2.29}
$$

The ELBO is now defined as follows:

$$
\mathcal{L}_{\theta,\phi}(x) := \mathbb{E}_{z\sim q}\left[\frac{p_\theta(x,z)}{q_\phi(z|x)}\right] = \log p_\theta(x) - D_{KL}(q_\phi(z|x) \parallel p_\theta(z|x)). \tag{2.30}
$$

Maximising the ELBO

$$
\hat{\theta}, \hat{\phi} = \arg\max_{\theta,\phi} \mathcal{L}_{\theta,\phi}(x) \tag{2.31}
$$

is equivalent to maximising the reconstruction error $\log p_\theta(x)$ and minimising the $D_{KL}(q_\phi(z|x) \parallel p_\theta(z|x))$. Namely, maximizing the observed data's log-likelihood and minimizing the approximate posterior divergence. The ELBO is a lower bound on the log-likelihood, hence the name 'Evidence Lower Bound'. The objective is to maximize the ELBO for the parameters of the model, which will in turn maximize the likelihood of the data. The KL divergence term in the ELBO acts as a regularizer, preventing the approximate posterior from collapsing to the prior distribution. The training process of VAE consists of maximizing the ELBO using gradient descent. By maximizing the ELBO, it is possible to learn the generative and inference models that can generate new samples from the data distribution and can also provide a compact representation of the data in the latent space.

2.6.3 Reparametrization Trick

The reparametrization trick is a technique used in Variational Autoencoders to make the optimization of the lower bound of the likelihood, also known as the Evidence Lower Bound (ELBO), more stable and efficient. The idea is to introduce a new random variable that we can sample from, which we will use to parameterize the generative process. This allows us to use backpropagation to optimize the parameters of the generative model.

The key idea is to reparameterize the latent variable z by introducing a new random variable ϵ and using it to compute z by:

$$
z = \mu + \sigma \odot \epsilon \tag{2.32}
$$

Where μ and σ are the learned parameters of the approximate posterior distribution and $\odot$ denotes element-wise multiplication.

The random variable ϵ is usually chosen to be a standard normal distribution. This way, the gradients to the parameters of the generative model can be computed by back-propagation, since the sampling operation is now differentiable. This allows us to optimize the parameters of the generative model in an end-to-end manner using gradient-based optimization. The reparametrization trick makes the optimization of the ELBO more stable and efficient, because of this, it is now a standard technique used in VAEs. The schematic representation of the process is shown in Figure 2.13. For the choice of standard normal $p(z) = \mathcal{N}(0, I)$, the KL divergence term has the following analytical form

$$D_{KL}(\mu, \sigma) = -\frac{1}{2}\sum_{i=1}^{n}(\log(\sigma_i^2) - \sigma_i^2 - \mu_i^2 + 1), \tag{2.33}$$

where σ_i and μ_i are the outputs of the encoder for a given x.

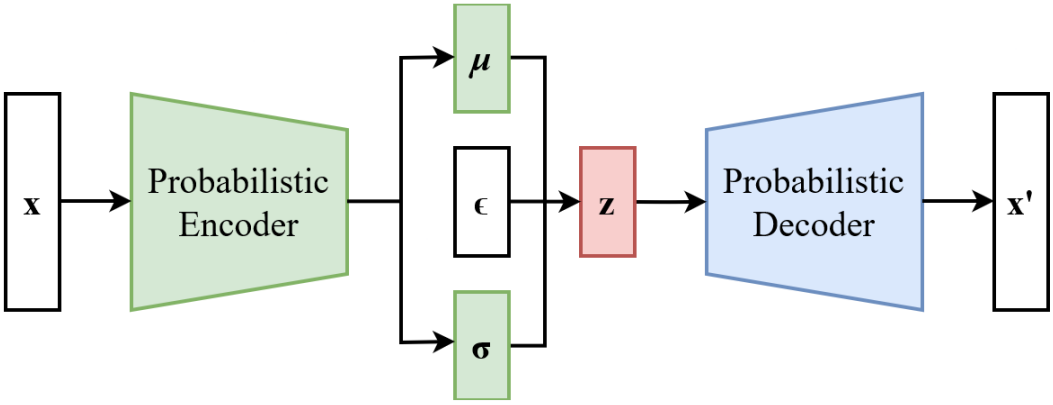


Figure 2.13: The variational autoencoder's design implements the reparameterization trick.

2.6.4 An Illustrative Example of VAE in HEP

An example of using a Variational Autoencoder for anomaly detection is in identifying rare particle interaction [65, 84]. In this scenario, a VAE can be trained on a dataset of known normal interactions, to learn the underlying distribution of the data. Once the VAE is trained, it can be used to generate new interactions from the learned distribution, by sampling from the latent space. The key idea is that the normal interactions should be consistent with the learned distribution and therefore can be reconstructed by the decoder with a low error. On the other hand, the anomalous interactions, which don't conform to the learned distribution, are likely to have a higher reconstruction error. Therefore, the VAE can be used to detect anomalous interactions by measuring the reconstruction error of the new interactions and comparing it to a threshold. Interactions with a reconstruction error above the threshold are considered anomalous and are likely to be rare particle interactions. Additionally, the VAE can be used to generate synthetic data to perform simulations. A common approach is to use VAE to generate synthetic events, which are used to mimic real events and train classifiers in an unsupervised way. In this case, by generating simulated events that closely resemble real-world events, the VAE can help to improve the performance of the classifiers, especially when working with small amounts

of real data. This is just one example of how a VAE can be used for anomaly detection in HEP, but it illustrates how this technique can be applied to identify rare events and improve our understanding of the underlying physical processes.

2.6.5 A bridge between Physics and Deep Learning

The Variational Autoencoder intuition was proposed by Diederik Kingma and Max Welling in 2013. The key innovation of VAEs is the use of variational inference, a technique from Bayesian statistics, to learn the latent representation. Variational inference allows the model to learn a probabilistic distribution over the latent space, rather than a single point estimate.

Kingma and Welling used concepts from several fields to develop the VAE model. The idea of using a partition function from statistical mechanics to calculate the probability distribution over the possible states of a physical system is used in VAE to calculate the probability distribution over the possible states of the data, represented by the latent variables. Additionally, the VAE objective function, expressed in equation 2.30, is closely related to the principle of Maximum Entropy, which states that the probability distribution that best represents the current state of knowledge is the one with the highest entropy. The VAE tries to find the probability distribution that maximizes the likelihood of the data and at the same time maximizes the entropy of the latent variables. Furthermore, the VAE objective function is also related to the Evidence Lower Bound (ELBO), which is a measure of the information contained in the data and the Mutual Information (MI), which is a measure of the amount of information shared between two variables.

VAEs have been applied in various fields, such as image [85] and video [86] generation, text generation [87], speech synthesis [88], and even drug design [89] and protein folding [90]. They are also used in fields such as computer vision, natural language processing, and generative design. Overall, Kingma and Welling used a combination of concepts from physics, Bayesian statistics, and information theory to develop VAE as a way to improve the understanding of the representations learned by DNNs and make the models more flexible and powerful.

2.7 Implementing a Neural Network

This section of the thesis will focus on the implementation of DL models using TensorFlow and Keras, two popular open-source libraries for building and training neural networks.

2.7.1 TensorFlow

TensorFlow [91] is an open-source software library for dataflow and differentiable programming across a range of tasks. It is a powerful tool for building and deploying machine learning models, particularly deep neural networks. TensorFlow was developed by the Google Brain team [92] and is now maintained by the TensorFlow team at Google. The core functionality of TensorFlow is its ability to perform computations on data using a dataflow graph. A dataflow graph is a directed acyclic graph (DAG) where the nodes represent mathematical operations and the edges represent the data flowing between the operations. This allows TensorFlow to perform complex computations in a highly optimized and efficient manner, making it well-suited for large-scale machine learning tasks,

and allows for the parallel execution of computations on large clusters of machines. One of the key features of TensorFlow is its support for a wide range of data types and platforms. It can be used to perform computations on a variety of platforms including CPUs, GPUs, and Tensor Processing Units (TPUs) [93]. TensorFlow also provides a variety of pre-built models and libraries, such as the TensorFlow Hub and the TensorFlow Model Garden, which can be used for common ML tasks like image classification, object detection, and natural language processing. TensorFlow also provides a variety of tools and libraries for building, training, and deploying machine learning models. TensorFlow is subdivided in various modules, these include:

- The **TensorFlow Core** API: this is the low-level API that provides the building blocks for building models with TensorFlow. This API provides some of the essential tools for building Machine Learning models, which enable us to create and manipulate tensors (multi-dimensional arrays), perform mathematical operations on them, and create and train models.
- The **Keras** API: a high-level API built on top of TensorFlow that allows for fast prototyping and experimentation with different neural network architectures.
- The **Estimator** API: another high-level API that simplifies the process of training and deploying machine learning models. It includes pre-built models for common machine learning tasks and provides an easy-to-use interface for training and deploying models on different platforms.
- **TensorFlow Serving**: this is a flexible, high-performance serving system for deploying machine learning models designed for production environments. TensorFlow Serving makes it easy to deploy new algorithms and experiments while keeping the same server architecture and APIs. TensorFlow Serving provides out-of-the-box integration with TensorFlow models but can be easily extended to serve other types of models and data.

TensorFlow also provides a variety of tools for deploying machine learning models in a production environment, on mobile devices and web browsers, thanks to TensorFlow Lite and TensorFlow.js respectively. TensorFlow provides support for various areas of artificial intelligence, such as computer vision, natural language processing, and reinforcement learning. TensorFlow also has a large and active community, which provides additional resources such as tutorials, pre-trained models and more.

2.7.2 Keras

Keras[2] is an open-source high-level deep learning API library, written in Python, which runs on top of TensorFlow. It was developed to make it easier to build, train, and test deep learning models. This package can abstract the complexities of making operations with TensorFlow and allows developers to work with a more user-friendly API for building and training models. It is designed to be user-friendly and modular, making it easy to create complex neural network architectures. Keras is built on top of other lower-level libraries such as TensorFlow, CNTK , or Theano [94], and it can run on top of those libraries or use them as a backend. It also supports multiple backends, including TensorFlow, making it easy to switch between them.

[2] https://github.com/keras-team/keras

One of the main advantages of Keras is its simplicity and ease of use. The API is intuitive and easy to understand, making it accessible to both beginners and experts. This package also has a large and active community, which provides a wealth of resources and tutorials for building and training models. The library also has a wide range of pre-built layers, optimizers, and metrics, making it easy to build models with complex architectures. Additionally, Keras provides support for a wide range of datasets and preprocessing tools, allowing users to quickly and easily load and prepare their data. Keras is highly extensible and allows developers to create custom layers, optimizers, and metrics. This makes it easy to implement new and experimental architectures and adapt existing models to new tasks. It also has built-in support for distributed training, which allows models to be trained on multiple GPUs or across multiple machines. This makes it easy to scale up training and improve performance on large datasets.

2.8 Model Compression

Deep learning models have become increasingly popular in recent years, but their large size can be a hindrance to their deployment in edge devices with limited computational resources. To address this issue, model compression techniques have been developed to reduce the size and computational complexity of these models without compromising their accuracy. This section, focus on two such techniques: weight pruning, quantization and knowledge distillation. Both of these techniques are effective in reducing the size of deep learning models while preserving their accuracy.

2.8.1 Pruning

Weight pruning is a technique used to compress deep learning models via the elimination of unnecessary values in the weight tensor, by practically setting the parameters' values to zero, which means cutting connections between nodes. The pruning is done during the training process to allow the NN to adapt to the changes. The TensorFlow Sparsity Pruning API [95] performed this optimization. The algorithm works by removing connections based on their magnitude during training, a graphical representation of the process can be seen in Figure 2.14. Therefore, a final target sparsity, i.e. target percentage of weights equal to zero is specified, along with a schedule to perform the pruning. As training proceeds, the pruning routing is scheduled to execute, removing the weights with the lowest magnitude, until the current sparsity target is reached.

There are different ways to implement weight pruning, but a common approach is to use a *magnitude-based pruning* criterion, where the magnitude of each weight is computed and the smallest weights are removed. The pruning threshold can be set manually, or determined through a heuristic or an optimization process. One way to prune weights is by using a *weight threshold*. In this method, a threshold is chosen and all weights below this threshold are set to zero. Another way is using a *percentage threshold*, where a certain percentage of the smallest weights is set to zero. Another approach is *iterative pruning*, where the model is trained, then pruned, retrained, and pruned again in a loop until the desired level of sparsity is reached. Another method is called *gradient-based pruning* which involves analyzing the gradients of the weights during the training process. Weights with small gradients are considered less important and are pruned. The last method is called *structured pruning* which involves pruning entire filters or neurons

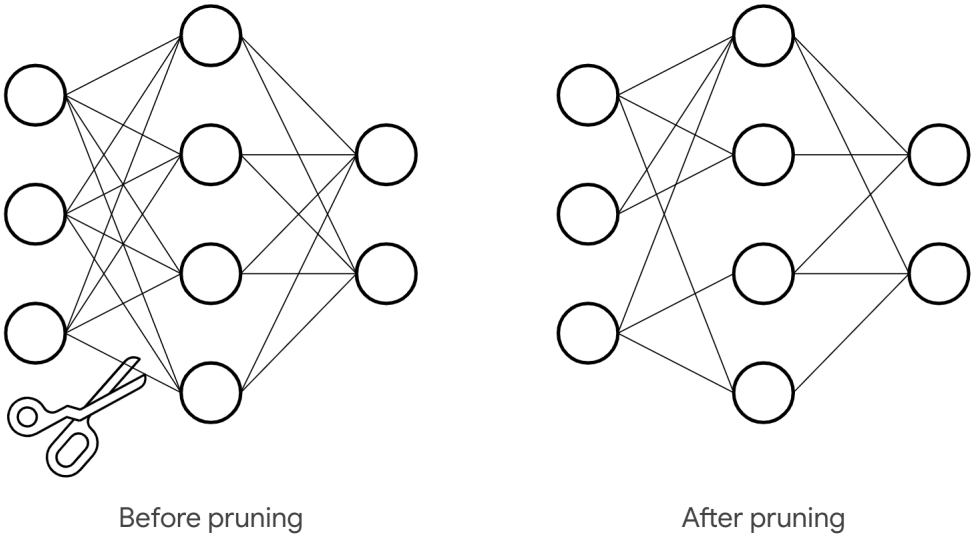


Figure 2.14: Graphical representation of the weight pruning optimization for FFNNs layer.

instead of individual weights. This method can be more efficient as it reduces the number of parameters much more than weight pruning.

It is important to note that after pruning, the model's architecture needs to be modified, to remove the pruned weights from the layers. Also, pruning may affect the model's performance, depending on the pruning method and the dataset. Therefore, it's important to evaluate the model's performance after pruning, and fine-tune or retrain the model if necessary. Weight pruning is a powerful technique for model compression, as it can significantly reduce the number of parameters in a model without significantly degrading its performance. It can also be used in combination with other compression techniques, such as quantization, to further reduce the model's size and computational cost.

2.8.2 Post-Training Quantization

Quantization is a process of converting continuous values into a limited set of discrete values. It refers to the process of convertion of the arithmetic used within the NN from high-precision floating-points to normalized low-precision integers (fixed-point), is an essential step for efficient deployment. This is usually done to reduce the memory and computational cost of a neural network, especially when deploying the network on resource-constrained devices such as mobile phones and embedded systems. The primary motivation for quantization is to reduce the memory footprint of a neural network. Floating-point values require more memory than fixed-point values because they need to store the sign, exponent, and mantissa. By reducing the number of bits used to represent the weights and activations, quantization reduces the memory requirements of a neural network, making it possible to deploy it on devices with limited memory. Another benefit of quantization is that it can significantly reduce the computational cost of a neural network. Fixed-point operations are faster than floating-point operations because they are more hardware-friendly, and can be implemented with simpler circuits. Furthermore, hardware accelerators such as GPUs and TPUs can perform fixed-point operations more efficiently than floating-point operations, leading to further reductions in computational cost.

Fixed-point numbers consist of two parts, integer and fractional as shown in Figure

2.15. Compared to floating-point, fixed-point representation maintains the decimal point within a fixed position, allowing for a more straightforward arithmetic operations.

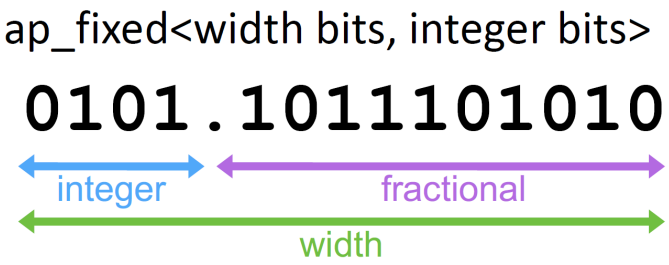


Figure 2.15: The representation of numbers with a fixed number of digits to the right and left of the decimal point is known as fixed-point number representation. In the hls4ml library (see section 2.9.4), the input, output, and parameters of the neural network model are associated with a C type called `fixed<width, integer>` that uses fixed-point number representation. Here, `width` represents the total number of bits used to represent the number, and `integer` represents the number of bits used to represent the integer part of the number. The use of fixed-point number representation ensures that the model is implemented with fixed hardware resources, which can lead to faster and more efficient execution.

Quantization is used to reduce the precision of weights and activations to a fixed number of bits. For example, instead of using 32-bit floating-point values to represent the weights and activations, a network might be quantized to use 8-bit fixed-point values. The number of bits used for quantization depends on the required accuracy and the memory constraints of the target deployment platform.

A possible technique widely used is the Post-Training Quantization (**PTQ**). It is used to reduce the memory and computational requirements of a machine learning model while maintaining its accuracy. In post-training quantization, the model's weights and activations are quantized, or represented with a smaller number of bits. The most common method of quantization is to represent the values with 8-bit integers (integer quantization) instead of 32-bit floating-point numbers. This reduces the memory requirements of the model by a factor of 4 and can also speed up computations on certain hardware, such as GPUs or TPUs.

The process of post-training quantization typically involves several steps:

1. **Calibration**: the model's weights and activations are passed through representative data (a subset of the training data) to generate a set of statistics, such as the minimum and maximum values for each layer. These statistics are used to determine the range of values that can be represented with the quantized data types.

2. **Quantization**: the weights and activations are then quantized to the nearest value within the determined range. This process can be done in different ways, such as symmetric or asymmetric quantization and per-channel quantization.

3. **Fine-tuning**: the quantized model is then fine-tuned by retraining on a small subset of the training data. This step helps to recover some of the accuracies that may be lost during quantization.

4. **Evaluation**: the quantized model is evaluated on a validation dataset to ensure that it has comparable accuracy to the original, non-quantized model.

One of the key benefits of post-training quantization is that it can be applied to any pre-trained model, regardless of its architecture or the framework it was trained with. Additionally, post-training quantization can be performed on the TensorFlow model, TensorFlow Lite model, or on the edge device, depending on the specific use case.

Additional points

- **Quantization methods**: Different methods of quantization can be used, such as symmetric or asymmetric quantization, per-channel quantization, or K-means quantization. *Symmetric quantization* maps the real numbers to integers by dividing the range of values into equal intervals, while *asymmetric quantization* maps the real numbers to integers by dividing the range of values into unequal intervals. *Per-channel quantization* is used to quantize the weights of convolutional layers separately for each channel, rather than globally for the entire layer. *K-means quantization* is a method that uses the k-means clustering algorithm to determine the optimal set of discrete values to represent the weights.
- **Quantization aware training**: It is an alternative to post-training quantization where the model is trained with the quantization process in mind, so the model learns to preserve accuracy even after quantization. This approach can lead to better accuracy compared to post-training quantization, especially for models with a large number of parameters.
- **Quantization granularity**: Post-training quantization can also be performed at different granularities, such as per layer or tensor. Per-layer quantization quantizes the activations and weights of each layer separately, while per-tensor quantization quantizes the activations and weights of each tensor in the model. The granularity of the quantization can be chosen based on the specific use case and the resources available on the target device.
- **Dynamic range**: Another important factor to consider when quantizing a model is the dynamic range of the values being quantized. The dynamic range is the ratio of the largest value to the smallest non-zero value in the data. A larger dynamic range requires more bits to represent the values accurately, so it's important to choose an appropriate quantization scheme that can represent the dynamic range of the data.

It is important to state that PTQ can lead to a small loss of accuracy compared to the original, non-quantized model, although recent techniques and research have mitigated this loss. It is also important to note that post-training quantization is typically only effective for reducing the memory requirements of the model, and may not have a significant impact on the performance of the model.

2.8.3 Quantization Aware of Training

Quantization-aware training (**QAT**) [96] is a method to train deep learning models while also quantizing the model parameters during training. This can be useful in scenarios where the model will be deployed on devices with limited computational resources, such as edge devices or mobile phones.

The process of QAT can be broken down into two main steps: quantization and fine-tuning. First, the model weights are quantized to a lower bandwidth, typically 8 bits. This is done by mapping each weight value to the closest representable value in the quantized space. After this, the model is fine-tuned using the quantized weights, to minimize the accuracy loss caused by the quantization. QAT can be performed using different types of quantization methods, such as uniform quantization or k-means quantization. Uniform quantization is the most common method, where the range of the weights is divided into a fixed number of intervals, and each interval is mapped to a discrete value. K-means quantization, on the other hand, is a more advanced method that clusters the weight values and maps each cluster to a discrete value. One of the challenges of QAT is dealing with the quantization errors introduced by the quantization process. To deal with these errors, Quantization-aware training technique computes the gradients for the quantized weights, rather than the full-precision weights. This allows the model to adjust its weights to better fit the quantized space, reducing the error caused by quantization. Another challenge of QAT is dealing with the increased memory and computation requirements caused by the fine-tuning step. This can be mitigated by using techniques such as weight pruning and knowledge distillation.

QKeras

QKeras [97] is a library for quantization-aware training of deep learning models that are built on top of the Keras (see section 2.7.2) framework. It provides a simple and easy-to-use interface for adding quantization-aware training to existing Keras models. As it has been underlined, the main idea behind quantization-aware training is to train a deep learning model with quantization in mind so that the model can be deployed on low-power and resource-constrained devices, such as smartphones and IoT devices, without a significant loss in accuracy. This is achieved by training the model with quantized weights and activations so that it can adapt to the quantization process and perform well on quantized hardware.

QKeras provides custom layers that can be used to replace the standard ones in a Keras model. These layers simulate the quantization process during training by applying quantization to the model. This allows the model to adapt to the quantization process and perform well on quantized hardware. Additionally, QKeras supports several different quantization schemes, including symmetric and asymmetric quantization, as well as different bit widths for the weights and activations. This allows for fine-tuning the quantization process to best suit the specific requirements of the model and the target hardware. QKeras also provides a set of pre-trained models that can be easily fine-tuned for specific tasks. This can greatly reduce the time and resources required to train a new model from scratch. In addition, it provides utilities for quantizing and deploying the trained model on different hardware platforms, such as Tensorflow Lite or TensorRT.

QKeras also uses a quantization-aware training loss function, which is calculated based on the difference between the quantized weights and activations and the original weights and activations. By using QKeras, developers can easily add quantization-aware training to their existing Keras models with minimal changes to their code. This can greatly simplify the process of deploying deep learning models on resource-constrained devices and can help to improve the performance and resource efficiency of these models.

2.8.4 Knowledge distillation

Knowledge distillation [98, 99] is a technique used to transfer the knowledge learned by a complex model, known as the teacher model, to a smaller model, known as the student model. The goal is to improve the performance of the student model by having it mimic the behaviour of the teacher model while being more computationally efficient.

The basic idea behind knowledge distillation is to use the teacher model to generate soft targets (probability distributions over classes) for the training data, rather than using the one-hot encoded true labels. These soft targets are then used to train the student model, which tries to replicate the teacher's predictions. This is done by minimizing the difference between the student's predictions and the teacher's soft targets, typically using a cross-entropy loss.

The process of knowledge distillation is usually divided into two steps:

- **Teacher model training**: The teacher model is trained on the dataset using different problem specific approach;
- **Student model training**: The student model is trained to replicate the teacher's predictions by minimizing the difference between the student's predictions and the teacher's soft targets.

One of the key advantages of knowledge distillation is that it allows transferring the knowledge of a large and powerful model to a smaller and more efficient model. Furthermore, it can be used to improve the generalization ability of the model and to make it more robust to adversarial attacks. There are different variations of knowledge distillation, such as using different types of teacher models (e.g. ensemble of models, attention-based models), different types of student models (e.g. shallow, deep) and different ways of generating the soft targets (e.g. temperature scaling, attention transfer). Temperature scaling is a popular method to generate soft targets, it involves raising the logits of the teacher model to a high temperature and then normalizing them to produce a probability distribution. The attention transfer method is another method, it involves computing an attention map for the teacher model and using it to weigh the logits of the student model.

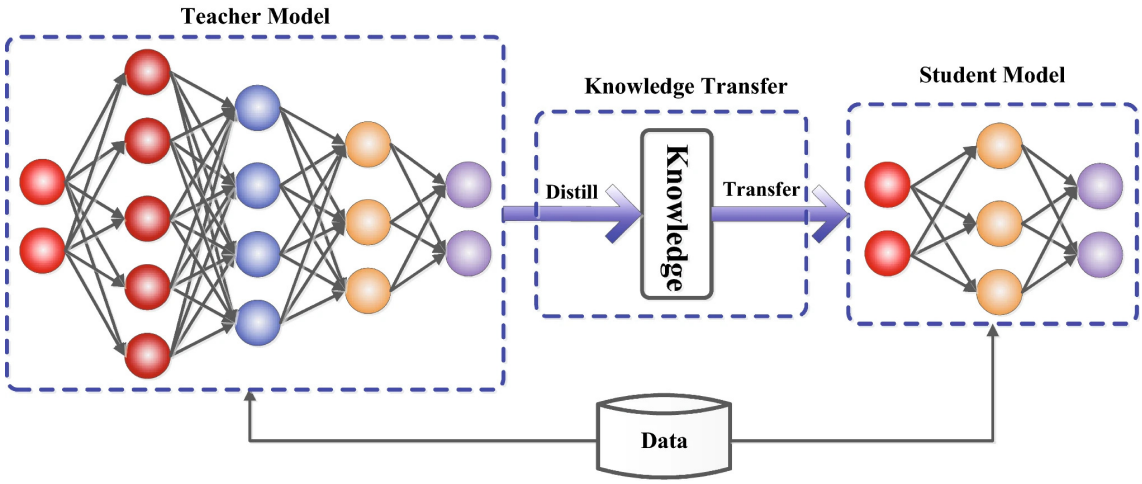


Figure 2.16: The standard teacher-student model for knowledge transfer [99].

There are several applications of knowledge distillation in HEP. One of the most common applications is in the area of particle classification and identification using

DNNs. For example, in the LHC experiment, particle collisions produce large amounts of data that need to be analyzed in real-time to detect interesting events. One of the main challenges is to identify the particles produced in the collision, as different particles have different physical properties and decay patterns. However, DNNs are typically large and computationally expensive, which makes them impractical for real-time analysis. Knowledge distillation can be used to address this issue by training the student model to mimic the behaviour of a larger and more accurate teacher model. The student model can then be used in place of the teacher model for real-time particle classification. Another application is in the area of jet tagging, which is the process of identifying the type of particle that initiated the jet. Jet tagging is an important task in HEP as it is used to identify the presence of new particles or to search for specific types of physics beyond the standard model. Similarly to particle classification, jet tagging can be performed using DNNs, but these networks can be large and computationally expensive. Knowledge distillation can be used to train a smaller and more efficient DNN for jet tagging. Additionally, Knowledge distillation can also be used to improve the performance of deep generative models for simulating the collider events. The student model can learn the underlying probability distribution of the simulations from the teacher model and generate events in real-time.

2.9 Model Acceleration

In recent years, there has been a growing interest in using heterogeneous computing architectures to accelerate machine learning models. One popular approach is to use Field-Programmable Gate Arrays (FPGAs) to accelerate model inference. FGPAs are a type of programmable logic device that can be configured to perform a wide range of digital functions. They have several advantages over traditional CPU and GPU architectures for machine learning, such as low power consumption and high flexibility. To facilitate the deployment of machine learning models on FPGAs, we will make use of the hls4ml library.

The goal of this section is to introduce the concept of heterogeneous computing and the use of FPGAs for model acceleration and to present the hls4ml library as a tool for fast inference on FPGA.

2.9.1 Heterogeneous Computing

Heterogeneous computing [100] refers to the use of multiple types of processing units in a single system, to exploit the unique strengths of each unit. In the context of deep learning, heterogeneous computing refers to the use of multiple types of electronic devices, such as Central Processing Units (CPUs), Graphics Processing Units (GPUs), and Field-Programmable Gate Arrays (FPGAs), to perform complex computations required by neural networks. Industry and High-Performance Computing (HPC) centres are successfully using heterogeneous computing platforms to achieve higher throughput and better energy efficiency by matching each job to the most appropriate architecture.

- One common type of processing unit found in heterogeneous systems is the central processing unit. **CPUs** are the traditional *brains* of a computer and are designed to handle a wide range of tasks, including general-purpose computing, data manipulation, decision-making, model training, and inference. They are typically composed of multiple cores, which can work in parallel to perform multiple tasks simultaneously. However, they are not as efficient as specialized processors like GPUs

or FPGAs in certain types of tasks, such as image and signal processing and may struggle with the high computational demands of deep learning tasks.

- Another type of processing unit found in heterogeneous systems is the graphics processing unit. **GPUs** are specialized processors designed for highly parallel computations required for rendering graphics, images and video processing. They have a large number of cores that can perform a large number of calculations simultaneously, making them well-suited for deep learning tasks. Additionally, many DL frameworks, such as TensorFlow and PyTorch, have built-in support for GPU acceleration, making it easy to harness the power of GPUs for deep learning tasks.

- A third type of electronic device that is becoming increasingly popular in heterogeneous systems is the field-programmable gate array. **FPGAs** are programmable chips that can be configured to perform specific tasks, such as signal processing or cryptography. They offer a high degree of flexibility and can be reprogrammed to adapt to changing workloads, which makes them well-suited for a wide range of applications, including edge computing, internet of things (IoT) devices, and data centres. They can be highly customized and optimized for specific tasks, such as image and signal processing, making them well-suited for deep learning tasks that require high levels of performance and low power consumption. Additionally, FPGAs can be reprogrammed to adapt to new tasks, making them a flexible option for deep learning applications.

In a heterogeneous computing system, different types of processing units can be used together to perform different tasks, depending on the specific requirements of the application. For example, a system might use a CPU to handle general-purpose computing tasks, while using a GPU to handle computationally-intensive tasks such as deep learning or image processing. Similarly, an FPGA might be used to perform specific tasks such as signal processing or cryptography, while the CPU and GPU handle other tasks. Each type of processing unit has its advantages and disadvantages, and the specific choice of processing unit will depend on the requirements of the application. For example, CPUs offer a high degree of flexibility and are well-suited for general-purpose computing tasks, while GPUs offer high performance for computationally-intensive tasks such as deep learning. FPGAs offer a high degree of flexibility and can be reprogrammed to adapt to changing workloads, making them well-suited for a wide range of applications.

In the context of HEP, such as the CMS experiment, heterogeneous computing is used to process and analyze the large amounts of data generated by the detectors [101]. This data is typically processed in stages, with different algorithms running on different types of hardware. For example, the initial trigger and data filtering may be performed on custom ASICs(application-specific integrated circuits) and FPGAs, while the final analysis may be performed on CPUs and GPUs. One of the challenges in using heterogeneous computing in HEP experiments is the need to efficiently transfer data between different types of devices and coordinate the execution of different algorithms. This is typically accomplished using a combination of specialized software frameworks and libraries, such as the CERN ROOT data analysis framework and the CUDA library for GPU computing. Deep learning algorithms, such as CNNs and RNNs, are increasingly being used in HEP experiments to improve the performance of data analysis algorithms. These algorithms are computationally intensive and require large amounts of memory, making GPUs well-

suited to parallel processing capabilities and or high-throughput data processing. In addition to GPUs, the CMS experiment also makes use of FPGAs for specific tasks such as trigger and data acquisition. They are well-suited for tasks such as data reduction and event reconstruction, as they can perform many calculations in parallel. This allows for a significant increase in processing speed compared to using only CPUs. In addition to GPUs, the CMS experiment also makes use of FPGAs for specific tasks such as trigger and data acquisition. FPGAs are reconfigurable devices that can be programmed to perform a specific set of tasks, making them well-suited for real-time data processing. They are also highly energy-efficient, which is important for large-scale data processing. FPGAs are also used in the context of the CMS experiment, to increase the speed of the data processing, filtering and compressing of the data (such as using AEs and VAEs) from the detectors, reducing the amount of data that needs to be transferred over the network and stored on disk. The CMS experiment also relies on a distributed computing infrastructure, where data is processed and analyzed on a network of distributed computing resources, including both CPU and GPU-based systems. These resources are connected through high-speed networks and can share data and processing tasks to perform complex data analysis tasks. Heterogeneous computing plays a critical role in the data processing and analysis of high energy physics experiments, allowing for the efficient processing of large amounts of data and the use of advanced machine learning algorithms.

2.9.2 Introduction to FPGAs

In the computer and electronics industry, there are two types of hardware used for performing computations: multipurpose hardware, such as consumer CPUs, which can be programmed to perform a wide range of tasks, and specialized hardware, such as Application-Specific Integrated Circuits (ASICs), which are designed and fabricated to perform a specific task and are highly optimized for that task. ASICs are permanently configured to perform only one application and require a multimillion-dollar design and fabrication effort. Computer software provides flexibility in changing applications and performing a wide range of tasks, but it is orders of magnitude worse in terms of performance, silicon area efficiency, and power usage compared to ASIC implementations.

Field Programmable Gate Arrays are devices that combine the benefits of both hardware and software, this makes them a popular choice for high-performance computing (HPC), digital signal processing, and other applications where flexibility is important. They implement circuits like hardware, providing huge power, area, and performance benefits over software, yet they can be reprogrammed cheaply and easily to implement a wide range of tasks. FPGAs implement computations spatially by simultaneously computing millions of operations in resources distributed across a silicon chip. Such systems can be hundreds of times faster than microprocessor-based designs. However, unlike in ASICs, these computations are programmed into the chip and not permanently frozen by the manufacturing process. This means that an FPGA-based system can be programmed and reprogrammed many times. But this flexibility comes at a cost: when producing an ASIC, the main expense is the building of the *master* with a much cheaper production line after development, while FPGAs have a much higher cost per chip than ASICs. Furthermore, an FPGA programmed to accomplish a specific task will perform worse than an ASIC designed with the same objective in mind. FPGAs are made up of replicated units of digital electronic circuits called logic blocks, embedded in a general routing structure, hence the gate and array in the name of this type of device. The logic blocks contain

processing elements for performing simple combinational logic, as well as flip-flops for implementing sequential logic. Any Boolean combinational function of five or six inputs can be implemented in each logic block. The general routing structure allows arbitrary wiring, so the logical elements can be connected in the desired manner. Modern FPGAs can compute functions on the order of millions of basic gates, working with clock frequencies in the hundreds of Megahertz. To boost speed and capacity, several FPGAs have additional ad-hoc elements embedded into the array, such as large memories, multipliers, and even microprocessors. With these predefined, fixed-logic units, FPGAs can implement complete systems in a single programmable device by configuring the connections between these units to perform complex algorithms.

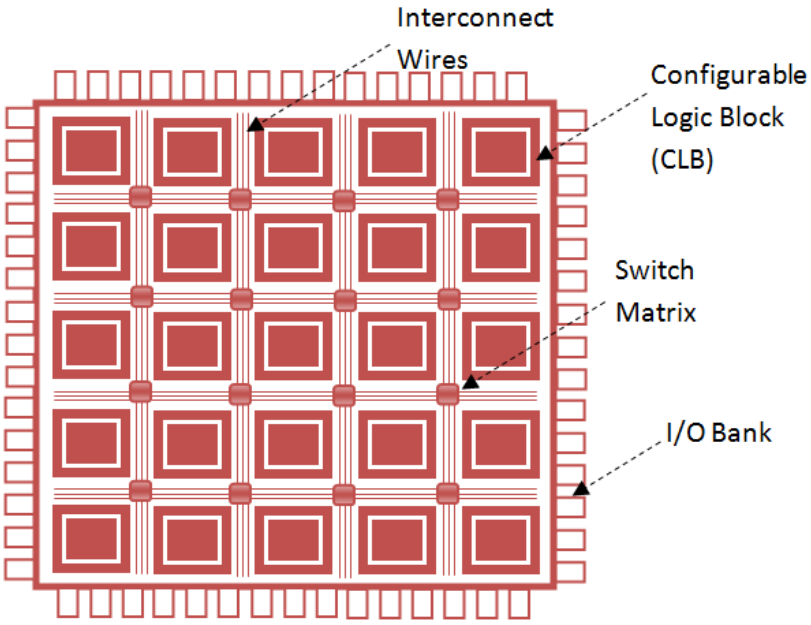


Figure 2.17: FPGA architecture with Configurable Logic Block (CLB) is shown in figure [102].

The FPGAs *electronic architecture* [102], shown in Figure 2.17 contains a large number of configurable logic blocks (CLBs) and programmable interconnects, which can be used to create complex logic circuits. The CLBs in an FPGA, shown in Figure 2.18, can be configured to perform a variety of logic functions, such as simple gates, complex digital logic functions, and memory elements. Each Configurable Logic Block consist of 2 slices. Those slices are further divided in 2 logic elements. Logic elements consist of:

- 4 input lookup Table (LUT);
- Full Adder and Mux logic;
- FlipFlop;

which can be used to implement any Boolean function of the inputs to the block. The 4 input Lookup table LUT is used to implement one of the following functionality:

- Combinational Logic design;
- Distributed RAM;
- Shift Register.

Also there are various dedicated circuits are present inside FPGA, such as

- Digital Clock Manager (DCM) is used to perform Clock Phase shift, De skew, Clock divider and frequency synthesis.
- Multiplier Block implement dedicated 18×18 multiplier with Signed and unsigned operation.
- Block RAM is a dedicated memory implement dual port 16kb memory.

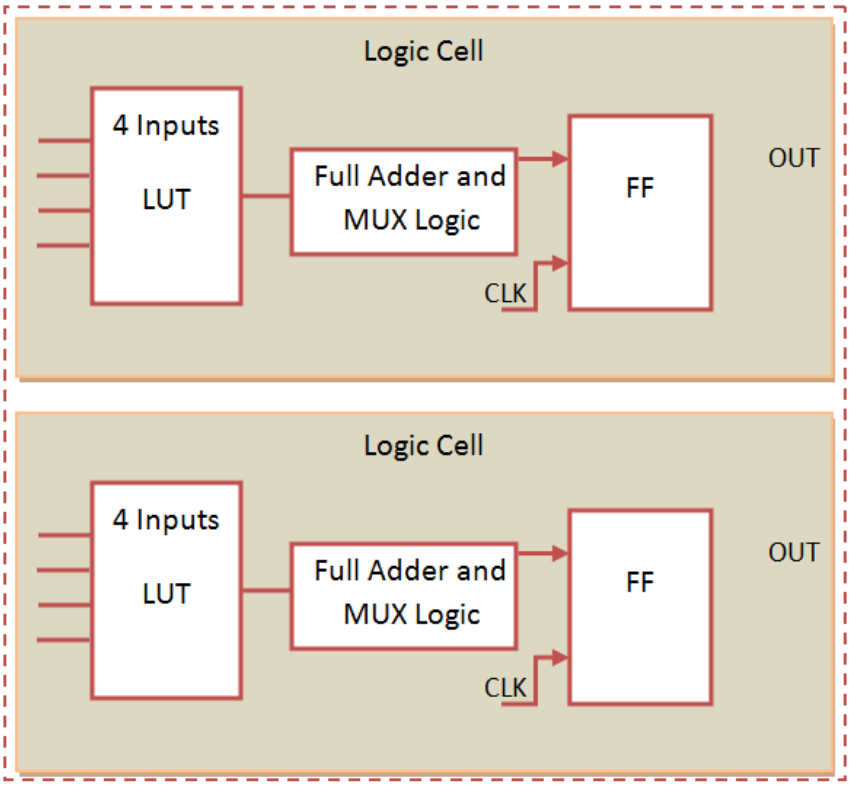


Figure 2.18: Configurable Logic Block [102].

The programmable interconnects are used to connect the CLBs together to create the desired logic circuit. They consist of a matrix of wires that can be programmed to connect any input or output of a CLB to any other input or output of a CLB. This flexibility allows for the creation of complex logic circuits with a high degree of customization and configurability. In recent years, FPGAs have become increasingly popular in the field of deep learning due to their high computational power and flexibility. They are well-suited for accelerating the inference of deep neural networks, such as CNNs and RNNs, using specialized hardware accelerators, such as digital signal processors (DSPs) and memory blocks. This allows for high-speed and low-latency processing of DL algorithms. One popular approach to using FPGAs for deep learning is to accelerate the matrix multiplications that are commonly used in deep learning. These multiplications can be implemented in parallel on the FPGA and GPU, resulting in a significant speedup compared to a CPU implementation. FPGAs can also be used to implement custom architectures for deep learning, such as custom layers or activation functions. This can further improve the performance of algorithms, as the custom architecture can be tailored to the specific requirements of the application. Additionally, FPGAs can be used to implement low-power and highly parallel deep learning architectures, and they can be used to offload

computation from the main CPU in embedded systems and edge devices where power and thermal constraints are a concern. Finally, FPGAs allow for easy scalability and reconfigurability, which is beneficial for deep learning applications that require frequent updates or changes in the network architecture.

2.9.3 Fast Inference on FPGAs

Fast inference for deep learning models on FPGAs [103] is a technique that allows for real-time processing of large amounts of data by leveraging the high-performance and low-power consumption characteristics of FPGA devices. This can be achieved through a process known as "compilation" or "synthesis," which converts a deep learning model, typically trained in software using frameworks such as TensorFlow or PyTorch, into a hardware implementation that can be run on an FPGA.

One way to achieve fast inference on FPGAs is by using High-Level Synthesis (HLS) tools such as HLS4ML, which can automatically generate VHDL or Verilog code for the FPGA based on a pre-trained machine learning model. This process can take advantage of the parallelism and regularity of deep learning operations, such as convolution and pooling, to map them onto the parallel processing elements of the FPGA. HLS tools can also optimize the memory accesses, data flow and parallelism of the design, by using techniques such as loop unrolling, pipelining and partitioning, which can improve the performance of the FPGA implementation.

Another way to achieve fast inference is through the use of pre-designed IP (Intellectual Property) blocks such as DSP (Digital Signal Processing) blocks, memory controllers, and high-speed transceivers that can be integrated into the FPGA. These IP blocks can be customized to the specific requirements of a deep learning model and can help to optimize the performance of the hardware implementation. DSP blocks can be used to accelerate the computation of convolution, matrix multiplication and activation functions, while memory controllers can be used to optimize the data transfer between the FPGA and memory. High-speed transceivers can be used to interface the FPGA with other devices such as cameras, sensors, and other FPGA or processors. In addition, FPGA-based deep learning acceleration can be improved by using specialized libraries and frameworks such as OpenCL, CUDA, or OpenVX. These tools allow the developer to program the FPGA in a high-level programming language such as C or C++, which can be easier and more efficient than working with low-level languages such as VHDL or Verilog. These libraries and frameworks can provide pre-designed functions and modules that can be used to accelerate the computation of the deep learning model and can also provide a higher-level abstraction of the FPGA architecture, which can simplify the design process. Finally, another technique to improve inference is through the use of quantization, which can reduce the precision of the model's weights and activations, thus reducing the memory and computation resources required by the model while maintaining its accuracy. A combination of these techniques can be used to achieve fast inference for deep learning models on FPGAs, such as using HLS tools, e.g. HLS4ML, pre-designed IP blocks, specialized libraries and frameworks, and quantization, to optimize the performance and efficiency of the hardware implementation.

High Level Synthesis

High-level synthesis (HLS) is a design process used to create digital circuits from high-level descriptions written in software programming languages, such as C, C++, or SystemC. The goal of HLS is to allow designers to focus on the algorithmic and architectural aspects of the design, rather than the low-level details of the hardware implementation. In this way, HLS enables designers to generate complex hardware designs faster and more efficiently than with traditional hardware description languages (HDLs), such as Verilog or VHDL. One of the key features of HLS is its ability to perform automatic optimizations. These optimizations include resource sharing, pipelining, loop unrolling, and many others. By automating these optimizations, HLS can create hardware designs that are much more efficient and faster than those designed manually. This is especially important in the context of modern digital circuits, which can have millions of gates and require complex optimization strategies to achieve high performance.

HLS is typically implemented as a toolchain that takes high-level code as input and generates a hardware description as output. The process consists of several stages, including parsing, high-level synthesis, scheduling, allocation, and verification. Each stage of the toolchain performs specific transformations on the input code to generate an optimized hardware description. One of the most popular HLS tools is the Vivado HLS from Xilinx. Vivado HLS allows designers to write C, C++, or SystemC code and generate RTL (register transfer level) code for FPGAs or ASICs. Vivado HLS supports a wide range of optimizations, including pipelining, loop unrolling, and memory optimization. It also provides advanced features, such as interface synthesis and hierarchical design. Vivado HLS is widely used in industry and academia for a variety of applications, including image and video processing, wireless communication, and deep learning. Another popular HLS tool is the LegUp HLS from the University of Toronto. LegUp HLS allows designers to write C code and generate FPGA hardware using LLVM (Low-Level Virtual Machine) compiler technology. LegUp HLS supports many optimizations, including resource sharing, loop pipelining, and automatic RTL generation. It also provides advanced features, such as interface generation and memory hierarchy management. LegUp HLS is widely used in research and education for a variety of applications, including scientific computing, signal processing, and computer vision.

2.9.4 HLS4ML package

HLS4ML (High-Level Synthesis for Machine Learning) [104] is an open-source tool that allows developers to translate neural networks [103] and boosted decision trees [105] into FPGA or ASICs firmware. The goal of hls4ml is to make it easy for developers to deploy machine learning models on embedded devices, such as drones, robots, and IoT devices, where power and computational resources may be limited.

The tool is implemented in Python and can be used with popular machine learning frameworks such as TensorFlow and Keras. It takes a pre-trained model and generates hardware-specific code, such as Verilog or VHDL, that can be used to program an FPGA or ASIC. The generated code can also be simulated using a software simulator to verify its functionality before it is deployed on the hardware. HLS4ML also includes several optimizations to improve the performance of the generated code, such as loop unrolling and pipelining. It also allows developers to specify constraints on the generated code, such as maximum clock frequency or resource utilization. One of the main benefits of

hls4ml is that it allows developers to take advantage of the high-performance and low-power characteristics of FPGAs and ASICs, while still being able to use the same machine learning models and frameworks they are familiar with.

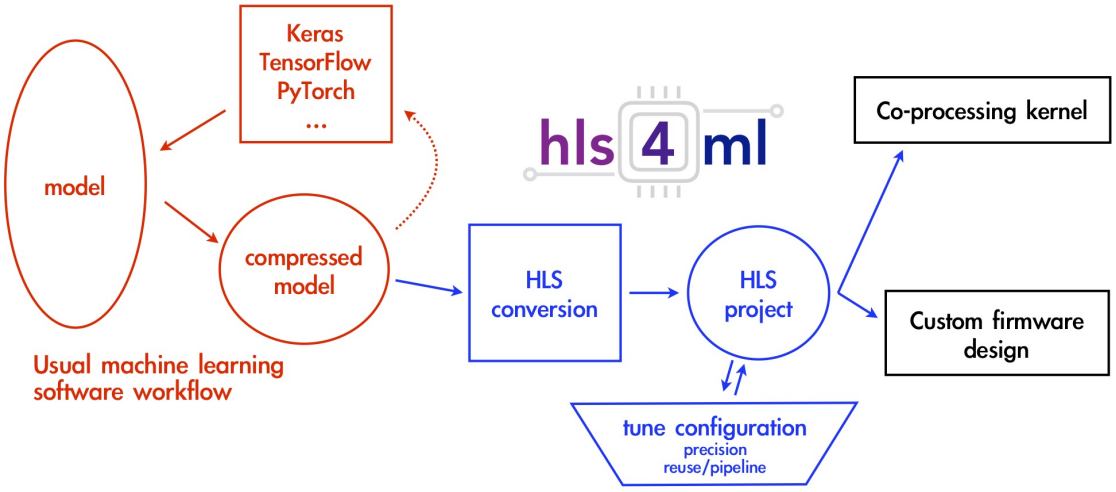


Figure 2.19: A typical workflow to translate a model into an FPGA implementation using hls4ml.

Members of the High Energy Physics community have developed hls4ml to translate ML algorithms into HLS code, enabling firmware development times to be drastically reduced. Figure 2.19 shows a schematic workflow. The goal of the hls4ml package is to empower a HEP physicist to accelerate ML algorithms using FPGAs, thanks to their tools for conversion. Indeed, hls4ml translates Python objects into HLS, and its synthesis automatic workflow, allows fast deployment times also for those who know how to write software or are not yet experts on FPGAs. Hardware used for real-time inference usually has limited computational capacity due to size constraints, and incorporating resource-intensive models without a loss in performance poses a challenge. One way to reduce a model's size is through post-training quantization, where model parameters are translated into lower precision equivalents. This process, by definition, is lossy and sacrifices model performance. With its interface to QKeras, hls4ml supports quantization-aware training [106], which makes it possible to drastically reduce the FPGA resource consumption while preserving accuracy. Through the use of this relatively new technique, hls4ml can integrate resource-intensive models without sacrificing performance and resulting in efficient interference. In this case, a fixed numerical representation is employed for the entire model, and the model is trained with this constraint during weight optimization. Using hls4ml we can compress neural networks to fit the limited resources of an FPGA.

In the context of the CMS experiment, hls4ml was used in the High-Level Trigger system to improve the event selection rate. A fully on-chip implementation of the machine learning model is used to stay within the $1\mu s$ latency budget imposed by a typical L1T system. Since there are several L1T algorithms deployed per FPGA, each of them should use much less than the available resources. Compared to the previous software-based version, the ML algorithm hardware implementation was able to process data way faster. This allowed for a higher event selection rate and improved overall performance of the HLT system. The use of hls4ml at CMS demonstrates the potential of the tool to accelerate machine learning applications in high-energy physics. The ability to convert pre-trained

models into hardware implementations on FPGAs can also open up new possibilities for other scientific fields such as astronomy and bioinformatics.

Chapter 3

Joint Representations in real-time

In the spirit of anomaly searches, the analysis carried out in this chapter focuses on the investigation of unsupervised learning of QCD and top jets using a neural network structure that is only trained on QCD jets. The aim is to examine the capability of the joint variational autoencoder model to encode class-related information in the latent space and how a more structured latent space geometry can enhance unsupervised classification.

3.1 Introduction

If new physics does exist at the scales investigated by the Large Hadron Collider, it is more elusive than expected [65, 107, 108]. Finding interesting results may be challenging using conventional methods, usually based on model-dependent hypothesis testing, without substantially increasing the number of analyses. Thus, standard signal-driven search strategies could fail in reaching new results, and unsupervised machine learning techniques could fill this critical gap. Such applications, running in the trigger system of the LHC experiments, could spot anomalous events that would otherwise go unnoticed, enhancing the LHC's scientific capabilities.

The most basic unsupervised machine learning technique is an autoencoder with a bottleneck [109]. It is constructed using a network that translates a high-dimensional data representation onto itself to create an average or typical entity. Standard autoencoders are recommended for unsupervised jet classification, but they are known to have problems in more general applications. The AE learns to compress and rebuild the training data very well, but when new, untrained data is run through the trained AE, it will produce a considerable loss or reconstruction error. By using the AE, it is possible to search for data that differs significantly from training data or even training data that is a small subclass of anomalous instances. In addition, the AE fails if the anomalous data is structurally simpler than the dominant class because the AE can encode simpler data with fewer features more efficiently. It is possible to overcome the disadvantages of AE by substituting a different classification measure for the reconstruction error. A possible alternative approach to the reconstruction error in the case of Variational Autoencoders [53] is to derive a metric from

the latent space embedding.

The analysis held in this final chapter of this thesis consists of implementing a JointVAE model [110], which is a variant of VAE for better latent representations, targeted at FPGA hardware architecture to determine the best latency and resource consumption without sacrificing model accuracy. Models will be optimized for classification between anomalous jets and QCD jets images, in an unsupervised setting by training solely on the QCD background. A comparison will be made between the reconstruction error and a latent space metric to determine the best anomaly detection score that enhances the separation of the two classes. The goal of the model is to reconstruct the input data information as accurately as possible. Additionally, because of the design of the JointVAEs architecture, the high-dimensional data representation is transformed into a compressed lower-dimensional latent distribution, both in continuous and discrete latent space, during the encoding stage. Subsequently, the decoder learns stochastic modelling and aims to generate input-like data by sampling from the latent distribution. The information about each dataset instance hidden in the high-dimensional input representation should be present in the latent space after training and the model can return the shape parameters describing the probability the density function of each input quantity given a point in the compressed space. With this application at the LHC, it could ideally be possible to classify the jets and even find anomalies using this latent space representation.

A companion tool based on High-Level Synthesis (HLS) named HLS4ML [104] will implement deep learning models in FPGAs, thanks also to the recent developments of the library [111] it has been possible to implement CNNs architectures for the machine vision task analysed in the following sections. Furthermore, compression by quantization of neural networks will reduce the model size, latency, and energy consumption [106]. We expect VAEs will find many uses in science, outperforming classical or standard deep learning baselines and even being able to solve challenges for physics beyond the Standard Model (BSM) that were previously unsolvable. Ideally applicable in a wide spectrum of signal background discrimination through anomaly detection, this application is expected to produce excellent results in a variety of fields.

3.2 Data Samples

In this chapter has been utilized the QCD and top jet samples generated for the top-tagging challenge described in Reference [112] for the following analysis, using the same jet image representation used in a previous AE study [65]. The jets are generated using Pythia8 [113] with a centre-of-mass energy of 14 TeV, without considering pile-up and multi-parton interactions. A fast detector simulation is performed using Delphes [114]. The jets are defined using the anti-k_T algorithm [115, 116] in FastJet [117] with a radius of $R = 0.8$. In each event, only the leading jet with p_T ranging from 550 to 650 GeV and $|\eta| < 2$ is kept, and the top jets are required to be matched to a parton-level top within the jet radius and all parton-level decay products to be within the jet radius. The jet constituents are defined using the Delphes energy flow algorithm and the top 200 constituents from each jet are used for analysis. The empty entries are zero-padded. We do not include particle ID or tracking information in our analysis. For pre-processing, we follow a similar procedure as described in References [65, 118]. The pre-processing is done at the constituent level before pixelization. The jet is first centred using the k_T-weighted centroid, such that the major principal axis points upwards. Then, the image is flipped in both axes so that the

majority of p_T is located in the lower left quadrant. Next, the image is pixelized into a 40×40 array, where the intensity is defined by the sum of p_T of the constituents per pixel. The pixel sizes are $[\Delta\eta, \Delta\phi] = [0.029, 0.035]$ and the image is cropped during pixelization to reduce the sparsity of information. Figure 3.1 shows the average of 10k QCD and top jet images after pre-processing. Our data consists of 200K QCD jets and 200K top jets, and for each analysis, the maximum number of jets possible is used. If equal numbers of QCD and top jets are required, 200K of each are used, and if the QCD jets are treated as the background, the full 200K QCD jets are kept and the number of top jets is varied. All results presented in this thesis use a 75/25 split of the data for training and testing. The data samples are shuffled and the testing data is selected randomly for each run.

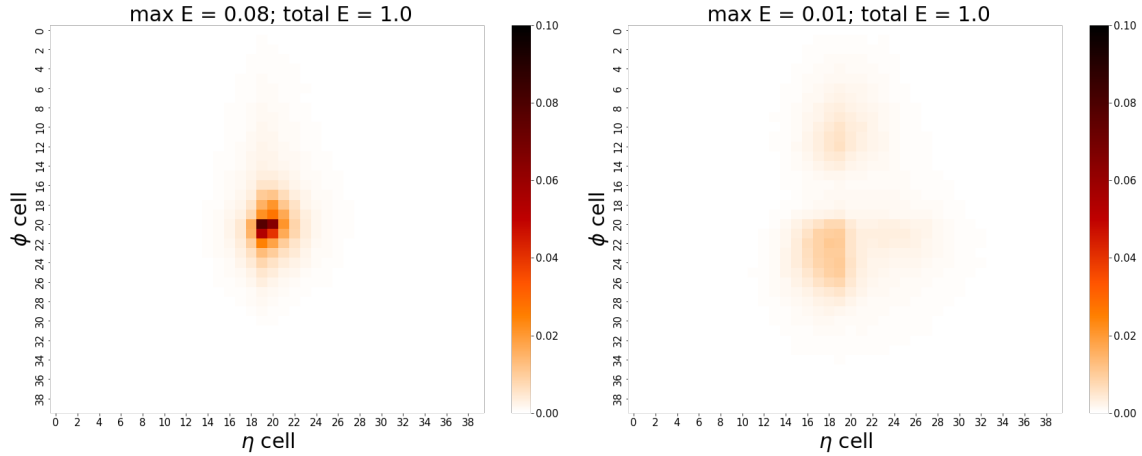


Figure 3.1: The mean of 200k pre-processed QCD and top jet images. (*Left*) Averaged on Calorimeter QCD jets and (*Right*) on Calorimeter tops tagged.

3.3 Choosing the model

This section it is described the specifics of the model implemented for the AD analysis and its training performance. Particular attention has been given to the Joint Variational Autoencoder (JointVAE) [110], which is a deep generative model that combines the strengths of Variational Autoencoders and discrete latent variable models. JointVAE models can handle both continuous and discrete data variables, which is a key advantage over traditional VAEs that are limited to only continuous latent variables. The JointVAE model consists of two main components: an encoder and a decoder. The encoder maps the input data to a latent space representation and the decoder maps the latent space representation back to the original input data. The encoder outputs the parameters of two different distributions: one for continuous latent variables and one for discrete latent variables. The reparameterization trick is used to sample values from these distributions. The discrete variables are modelled using a discrete distribution, such as the Gumbel-Softmax [119], while the continuous variables are modelled using a Gaussian distribution. The JointVAE model is trained by maximizing the likelihood of the input data given the latent representation. This is done by minimizing the difference between the generated data from the decoder and the original input data.

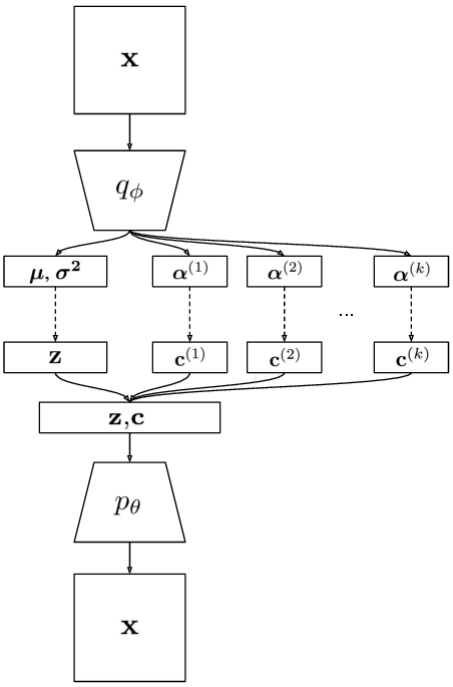


Figure 3.2: The JointVAE architecture takes input x and encodes it into parameters for latent distributions using the encoder q_ϕ. Samples are generated from these distributions using the reparameterization trick, as shown by the dashed arrows in the diagram. The generated samples are concatenated and then decoded through the decoder p_θ [110].

3.3.1 JointVAE Architecture and Operation

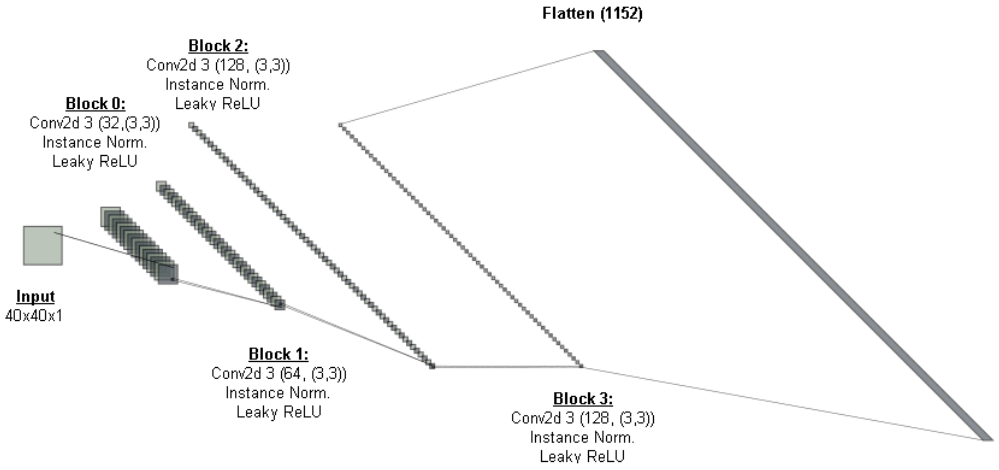
For the analysis, it is used a modification of the standard VAE framework, as described in section 2.6, allows us to model a joint distribution of the discrete latent variables in addition to the continuous ones pursued with the standard VAE. Letting z denote a set of continuous latent variables and c denote a set of categorical or discrete latent variables, we define a joint posterior $q_\phi(z, c|x)$, prior $p_\theta(z, c)$ and likelihood $p_\theta(x|z, c)$. Assuming the latent distribution is jointly continuous and discrete, and latent variables conditionally independent, we can rewrite the KL divergence term in the VAE loss equation 2.30 as:

$$\begin{aligned}
D_{KL}(q_\phi(z,c|x) \parallel p_\theta(z,c)) &= \mathbb{E}_{q_\phi(z,c|x)}\left[\log \frac{q_\phi(z,c|x)}{p_\theta(z,c)}\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\mathbb{E}_{q_\phi(c|x)}\left[\log \frac{q_\phi(z|x)q_\phi(c|x)}{p_\theta(z)p_\theta(c)}\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\mathbb{E}_{q_\phi(c|x)}\left[\log \frac{q_\phi(z|x)}{p_\theta(z)}\right] + \mathbb{E}_{q_\phi(z|x)}\mathbb{E}_{q_\phi(c|x)}\left[\log \frac{q_\phi(c|x)}{p_\theta(c)}\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{q_\phi(z|x)}{p_\theta(z)}\right] + \mathbb{E}_{q_\phi(c|x)}\left[\log \frac{q_\phi(c|x)}{p_\theta(c)}\right] \\
&= D_{KL}(q_\phi(z|x) \parallel p_\theta(z)) + D_{KL}(q_\phi(c|x) \parallel p_\theta(c)),
\end{aligned} \tag{3.1}$$

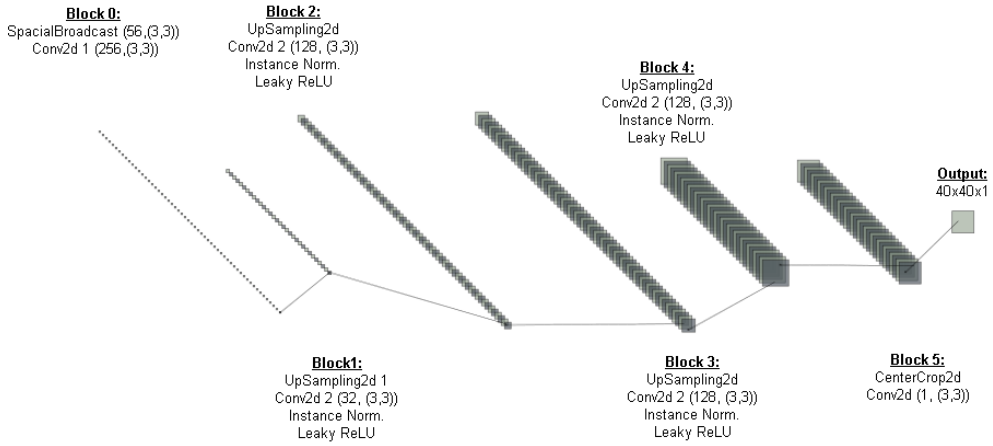
i.e. we can separate the discrete and continuous KL divergence terms. Under this assumption, the JointVAE loss finally becomes

$$\mathcal{L}_{\theta,\phi}(x) = \mathbb{E}_{q_\phi(z,c|x)}[logp_\theta(x|z,c)] - \beta D_{KL}(q_\phi(z|x) \parallel p_\theta(z)) - \beta D_{KL}(q_\phi(c|x) \parallel p_\theta(c)), \tag{3.2}$$

where β is a positive constant. When $\beta > 1$ it is theorized that the combination of the increased force exerted by the posterior $q_\phi(z|x)$ aligns with the prior $p_\theta(z)$, combined with maximizing the likelihood term, leads to efficient and well-separated representations of the data [120].



(a) Encoder q_ϕ.



(b) Decoder p_θ.

Figure 3.3: The neural network architecture. The figure only shows the encoder and decoder models without the intermediate `Dense` layers used to compute the mean, variance (32-dimensional), and categorical (20-dimensional). These layers receive input from the `Flatten` layer illustrated in Figure 3.3a. The output of these layers is then fed into the custom layer `joint sampling`, which performs the parametrization trick on continuous and discrete latent vectors. The latter vector is subsequently used as input to the decoder model, shown in Figure 3.3b.

The JointVAE model's final architecture is depicted in Figure 3.2. The encoder

implemented in the following analysis is depicted in Figure 3.3 It has been designed to output the parameters for both the continuous distribution, mean μ and variance σ^2, and the discrete distributions $\alpha^{(i)}$. Sampling from these distributions is achieved using the reparameterization trick, discussed in detail in the following subsection, and involves drawing a sample from the normal distribution $z \sim \mathcal{N}(\mu_i, \sigma_i^2)$ and the discrete distribution $c_i \sim g(\alpha^i)$. These samples, along with the continuous and discrete variables, are then combined and concatenated into a single latent vector, which serves as the input to the decoder, represented in Figure 3.3b.

3.3.2 Reparametrization of Latent Variables

In line with the traditional VAE setup, we parametrize $q_\phi(z|x)$ as a factorized Gaussian (see section 2.6.3), meaning $q_\phi(z|x) = \prod_i q_\phi(z_i|x)$ where $q_\phi(z_i|x) = \mathcal{N}(\mu_i, \sigma_i^2)$, and set the prior distribution as a standard Gaussian $p_\theta(z) = \mathcal{N}(0, I)$. Both μ and σ^2 are represented by neural networks.

The parameterization of $q_\phi(c|x)$ is more challenging. To make it differentiable from its parameters, a direct parametrization using categorical distributions is not possible. Some studies [119, 121] have proposed a differentiable solution using the Gumbel-Max trick. If c is a categorical variable with the class of categorical distribution $\alpha_1, ..., \alpha_k$, then we can sample from a continuous approximation of the categorical distribution by sampling $g_1, ..., g_k$ i.i.d. samples drawn from the continuous distribution $Gumbel(0, 1)$. This enables the use of backpropagation in training models that have discrete outputs, such as categorical distributions. We use the softmax function as a continuous, differentiable approximation of gradient descent, and generate **k-dimensional** sample vectors

$$y_i = \frac{\exp(\log(\alpha_i) + g_i)/\tau)}{\sum_{j=1}^{k} \exp((\log(\alpha_j) + g_j)/\tau)} \qquad \text{for} \quad i = 1, ..., k, \tag{3.3}$$

where τ is a temperature parameter which controls the amount of randomness in the samples and it can be decreased during training to make the approximation close to the true categorical distribution. As $\tau \to 0$ the softmax becomes an $\arg\max$ and the Gumbel-Softmax distribution becomes the categorical distribution. During training, we let $\tau > 0$ to allow gradients past the sample, then gradually anneal the temperature τ, but not completely to 0, as the gradients would blow up. The Gumbel-Softmax is a powerful tool for variational inference and generative models, as it enables the use of the reinforced gradient estimator for training models with discrete latent variables. The function is differentiable for the inputs, making it suitable for use in backpropagation, and it can be used as a continuous approximation of the $\arg\max$ function in discrete optimization problems. Additionally, the Gumbel-Softmax can be interpreted as a continuous relaxation of the one-hot encoding representation, enabling efficient and accurate optimization of models with discrete latent variables. We can model $q_\phi(z|x)$ by a combination of separate Gumbel-Softmax distributions, represented as $q_\phi(c|x) = \prod_i q_\phi(c_i|x)$ where $q_\phi(c_i|x) = g(\alpha^{(i)})$ is a Gumbel-Softmax distribution with class probabilities $\alpha^{(i)}$. The prior $p_\theta(c)$, is set to be an ensemble of uniform Gumbel-Softmax distributions, enabling the utilization of the reparametrization technique [53] and efficient training of the discrete model.

The KL divergence between the Gumbel-Softmax and the categorical distribution is given by:

$$D_{KL}(q_\phi(c|x) \parallel p_\theta(c)) = \sum_{j=1}^{J} \sum_{k=1}^{K_j} q_\phi(c_j = k|x) \log \frac{q_\phi(c_j = k|x)}{p_\theta(c_j = k) + \epsilon} \tag{3.4}$$

where J is the number of discrete latent variables, K_j is the number of categories for the j-th latent variable, $q_\theta(z_j = k|x)$ is the Gumbel-Softmax approximation of the posterior probability of the k-th category of the j-th latent variable given the input x, and $p(c_j = k)$ is the prior probability of the k-th category of the j-th latent variable. ϵ is a small constant (e.g. 1e-8) added to $p(c_j = k)$ to avoid dividing by zero. This helps to prevent numerical instability during training, which can lead to `nan` values and other optimization problems.

3.3.3 Training details

For the anomaly detection task using a JointVAE, it is important to carefully tune the hyperparameters to maximize performance. This is because the model needs to be able to effectively distinguish between normal and anomalous data points to accurately identify anomalies. However, at the same time, it is also important to avoid overfitting the model to the training data. Overfitting occurs when a model becomes too complex and starts to fit the noise in the training data rather than the underlying patterns, which can result in poor performance on new data, see section 2.1.2. To strike a balance between model performance and generalization, those hyperparameters were chosen for the JointVAE model:

- **Latent distribution**: The latent space is 32-dimensional for the continuous and one 20-dimensional discrete part.
- **Optimizer**: The model is trained using the AdamW optimizer [47] (see the subsection 2.1.4), which is a variant of the popular Adam [43] optimizer that adds weight decay to the parameter updates. The optimizer is configured with a decay rate $\lambda =$ 1e-3, which controls the amount of regularization applied to the model parameters. Additionally, the ReduceLROnPlateau method is implemented to adjust the learning rate when the validation total loss plateaus. Specifically, the learning rate is reduced by a factor of 0.5 if no improvement in the validation loss is observed after 3 epochs.
- **Batch size**: The model is trained using a batch size of 128. This means that the model updates its parameters after processing 128 samples from the training data at a time. This choice of batch size balances the trade-off between the efficiency of computation and the quality of the parameter updates.
- **Epochs**: The model is trained for 8 epochs, where each epoch corresponds to a full pass through the training data. This choice of epoch number is based on a trade-off between maximizing the model's performance and avoiding overfitting in the latent space. Early stopping is used if there is no improvement in the validation loss observed after four epochs.
- β: This is a hyperparameter used in the model's objective function eq. 3.2, which balances the reconstruction loss and the KL divergence terms. Specifically, β is set to 3e3, which indicates a preference for high-quality reconstructions with a moderate amount of compression in the latent space.

- τ: The temperature, in the Gumbel-Softmax distribution, is an important parameter for the JointVAE model as it affects the sampling of the discrete latent space. Tuning this parameter is crucial for generating quality samples and detecting anomalies. Our model performed best with a temperature of 50, determined through a systematic search over a range of values. This parameter appears in the equation 3.3.

- ϵ: The epsilon term is typically added to the KL divergence in the formula 3.4 between the Gumbel-Softmax distribution and a categorical distribution to ensure numerical stability during optimization. In the implementation, we chose the value of ϵ =1e-7.

In a JointVAE, the choice of the *latent dimension* for both the continuous and discrete latent space is a critical hyperparameter that can greatly affect the model's performance. Generally, larger latent dimensions provide the model with more flexibility to capture complex structures in the data, but can also increase the risk of overfitting, especially when the training data is limited. The selection of the latent dimension is generally done through trial and error, by training the model with different values and evaluating its performance on a validation set. In the case of the *continuous* latent space, a latent dimension that is too small may result in the loss of relevant information, while a dimension that is too large may lead to overfitting. A commonly used rule of thumb is to set the latent dimension to be smaller than the input dimension, but large enough to capture the essential information in the data. In the case of the *discrete* latent space, the choice of the dimension can be more challenging, as there are typically a limited number of discrete variables that can be used to represent the data. One approach is to use a smaller latent dimension and a richer set of discrete variables to compensate for the reduced expressive power. Another approach is to use a larger latent dimension and enforce sparsity constraints to avoid overfitting. In this analysis, we choose the first approach as it has reached better performance in the AD task. To avoid overfitting it is crucial to evaluate the model's performance on a validation set and avoid selecting the dimension based solely on the performance on the training set.

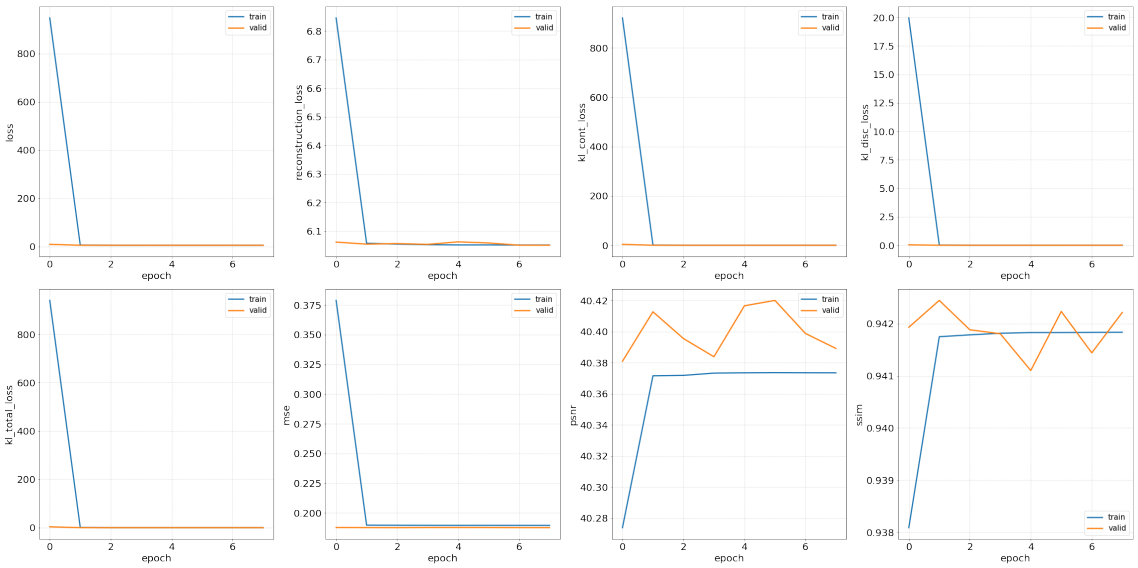


Figure 3.4: The model's performance is evaluated over the epochs using different metrics.

To provide a quantitative measure of how well the model can generate realistic samples that match the real data, while also ensuring that the learned latent distribution matches the

prior one, some metrics are commonly used. By monitoring these metrics during training, we can ensure that the model is learning the underlying data distribution and generating high-quality samples that are useful for anomaly detection. The metrics used to evaluate the training performance are:

1. **Total Loss**: is the sum of the reconstruction loss and the Kullback-Leibler divergence, which measures how well the generated samples match the real data and how well the latent distribution matches a prior distribution, respectively. The total loss is used to balance the trade-off between reconstruction accuracy and the encoding capacity of the model.

2. **Reconstruction Loss**: is a metric used to evaluate the quality of generated samples in a generative model. It measures the difference between the generated samples and the real data on a pixel-by-pixel basis, typically using a loss function such as binary cross-entropy (BCE), as used for the training here. In contrast to mean squared error (MSE), BCE is preferred when the pixel values are normalized between 0 and 1, as is often the case in image generation tasks. BCE is also less sensitive to outliers, making it a more robust choice for reconstruction loss in many scenarios. By using BCE as our reconstruction loss, we can ensure that our generative model is producing high-quality samples that closely resemble the real data. This is because BCE penalizes the model for discrepancies between the generated samples and the real data at the pixel level, effectively encouraging the model to learn accurate pixel distributions. Additionally, by avoiding the use of MSE, we can avoid potential issues with outliers or other anomalies in the data that may adversely affect the training process.

3. **KL Divergence** (Continuous and Discrete): The KL divergence is a metric used in VAE models to measure the difference between the learned latent and prior distributions. In JointVAE models, which utilize both continuous and discrete latent variables, there are two sources of KL divergence. The total KL divergence is the sum of the KL divergences for each type of variable, as shown in equation 3.1. Lower KL divergence values indicate a better match between the learned distribution and the prior. However, it is important to note that a perfect match (i.e., KL = 0) is a failure case for VAE models, as they require a structured latent space that differs from the prior distribution at some points to perform effective data generation. A nonzero KL divergence indicates that the model has learned meaningful structure in its latent space. In the case of JointVAE, the KL divergence plays a crucial role in *balancing* the contributions of the continuous and discrete latent variables. By minimizing the KL divergence, the model can effectively learn the relationships between the different components of the latent space, resulting in high-quality data generation.

   For the continuous KL function, we implemented the formula 2.33. The discrete KL function implemented in our code computes the KL divergence between the Gumbel-Softmax distribution and a categorical distribution for the discrete latent variables in the JointVAE model. The categorical codes are first converted into probabilities using the softmax function. The KL divergence is then calculated using the formula:

$$D_{KL}(q_\phi(c|x) \parallel p_\theta(c)) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{K} q_\phi(c_{i,j}|x)\log\frac{q_\phi(c_{i,j}|x)}{p_\theta(c_j)} \tag{3.5}$$

where $q_\theta(c_{i,j}|x)$ is the probability of the j-th category for the i-th sample given the input x, $p(c_j)$ is the prior probability of the j-th category, and N is the number of samples. To calculate the KL divergence efficiently, the entropy of the logits is first computed using $h_1 = q_p \cdot \log(q_p + \epsilon)$, where q_p is the probability distribution obtained from the softmax function and ϵ is a small constant for numerical stability. The cross-entropy with the categorical distribution is then calculated using $h_2 = q_p \cdot \log(1/K + \epsilon)$, where K is the number of categories. Finally, the KL divergence is obtained by taking the difference between h_1 and h_2, summing over the categories, and taking the mean over the samples. The resulting KL divergence is multiplied by a hyperparameter β and averaged over the batch to obtain the final loss.

4. **Peak Signal-to-Noise Ratio** (PSNR): measures the quality of the generated images by computing the ratio between the maximum possible pixel value and the mean squared error (MSE) between the generated and real images. Higher PSNR indicates better quality of generated images.

5. **Structural Similarity Index** (SSIM): this metric measures the similarity between the generated and real images in terms of luminance, contrast, and structure. Higher SSIM indicates better quality of generated images.

6. **Mean Squared Error** (MSE): This measures the average squared difference between the generated and real images. Lower MSE indicates better quality of generated images.

	LOSS	BCE	KL-cont	KL-disc	MSE	PSNR	SSIM
Train	6.053	6.051	4.240e-04	2.827e-04	0.190	40.374	0.942
Validation	6.061	6.060	4.470e-04	4.470e-04	0.188	40.427	0.942

Table 3.1: Final results after 8 epochs of the evaluation metrics for training performance.

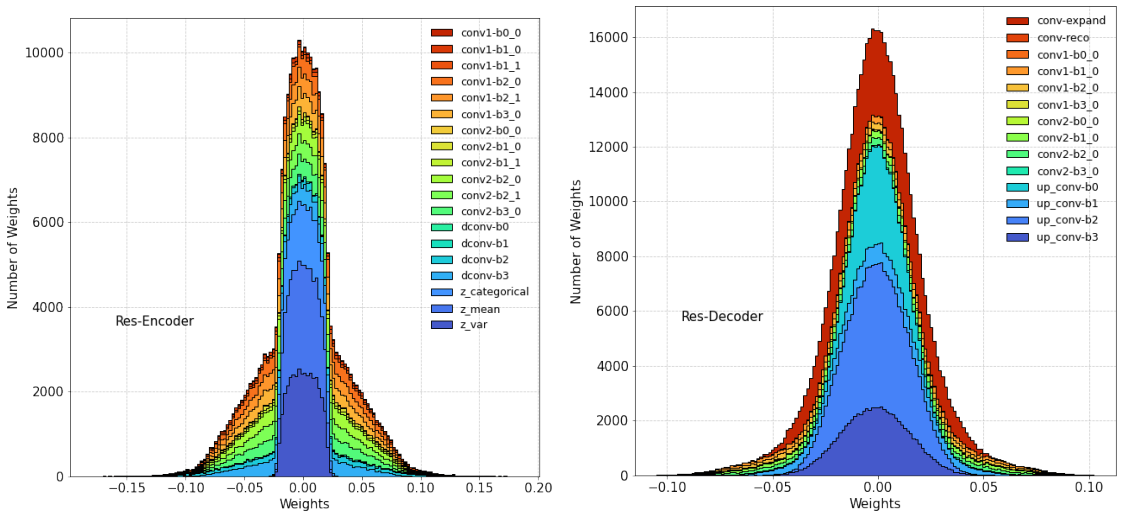
The training performance is depicted in Figure 3.4 and in Table 3.1 are shown the final results of the metrics used to evaluate the performance of the model's training for both the train and validation subset of QCD datasets images. The model is trained on NVIDIA-SMI T4 GPU, provided for free by Google Colab [122].

3.3.4 Weight Distributions

In DL models, the weights are the parameters that are learned during the training process. These weights are responsible for making predictions on unseen data. Initializing the weights with a specific distribution is an important factor in determining the performance of the model. In the implemented JointVAE model, `he_normal` is used to initialize the weights. *He normal* [123] is a variation of the popularly used *Xavier normal*[1] [124]

[1]It is named after its creator, Xavier Glorot, and is designed to ensure that the variance of the outputs from each layer in a network is approximately equal to the variance of its inputs. This helps prevent the variance from exploding or vanishing as it propagates through the network during training. In Xavier's normal initialization, the weights of each layer are randomly initialized from a normal distribution with

initialization, with a slight modification to better suit the initialization of deep networks with ReLU [68] and Leaky ReLU activation [69].



(a) Encoder weights.

(b) Decoder weights.

Figure 3.5: The weights per layer for the encoder and the decoder at Floating-point precision.

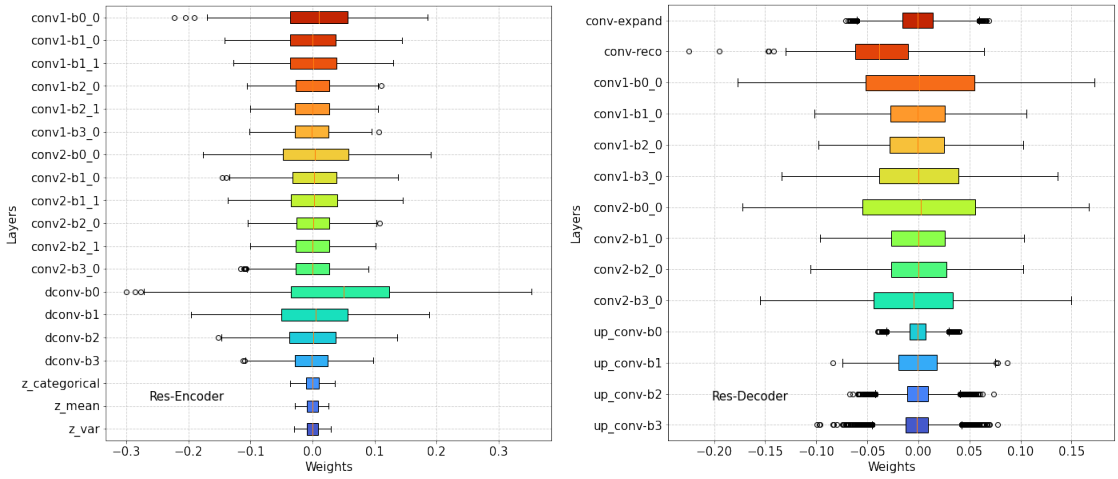
He normal initialization is a Gaussian distribution with a mean of 0 and a standard deviation of $\sqrt{\frac{2}{n}}$, where n is the number of input neurons for a given layer. The AdamW is used as a method to prevent overfitting by adding a penalty term to the loss function. The penalty term encourages the model to have smaller weights, reducing the complexity of the model and avoiding overfitting. In the JointVAE model, a weight decay of 0.001 is used. The penalty term is equal to 0.001 times the sum of the squares of all weights in the model, which is added to the loss function. By using He normal initialization and weight decay, the JointVAE model can have a better chance of finding a good solution during the training process, while avoiding overfitting on the training data. The choice of initializing the weights and using weight decay is problem-specific and may require some tuning to obtain the best results. In Figure 3.5 are visible the gaussian shape caused by he normal initialization and the small magnitude values of weights due to the weights decay.

Whisker Plots

A whisker plot [125] is a type of box plot that is often used to visualize the distribution of a set of data. It consists of a box that represents the interquartile range (IQR), which covers the middle 50% of the data and two whiskers that extend from either side of the box to represent the range of the remaining data. Outliers are plotted as individual points beyond the whiskers. Whisker plots are particularly useful in comparing the distribution of different sets of data, such as the comparison of the layer weights of a JointVAE with normal layers and a JointVAE with quantized layers. By comparing the box and whiskers of the two plots, we can see if the distribution of layer weights in the quantized model is significantly different from the distribution of layer weights in the normal model. In the case of a JointVAE with quantized layers, we would expect to see a shift in the distribution

mean 0 and variance $\frac{1}{N}$, where N is the number of inputs to the layer. This helps ensure that the weights are initially set to reasonable values and that the output variance is not too large or too small.

of layer weights, as quantization can introduce quantization errors and cause the weights to differ from their original values.



(a) Encoder layer weights.

(b) Decoder layer weights.

Figure 3.6: Layer weight numerical values for the floating-point model for a subset of the test data.

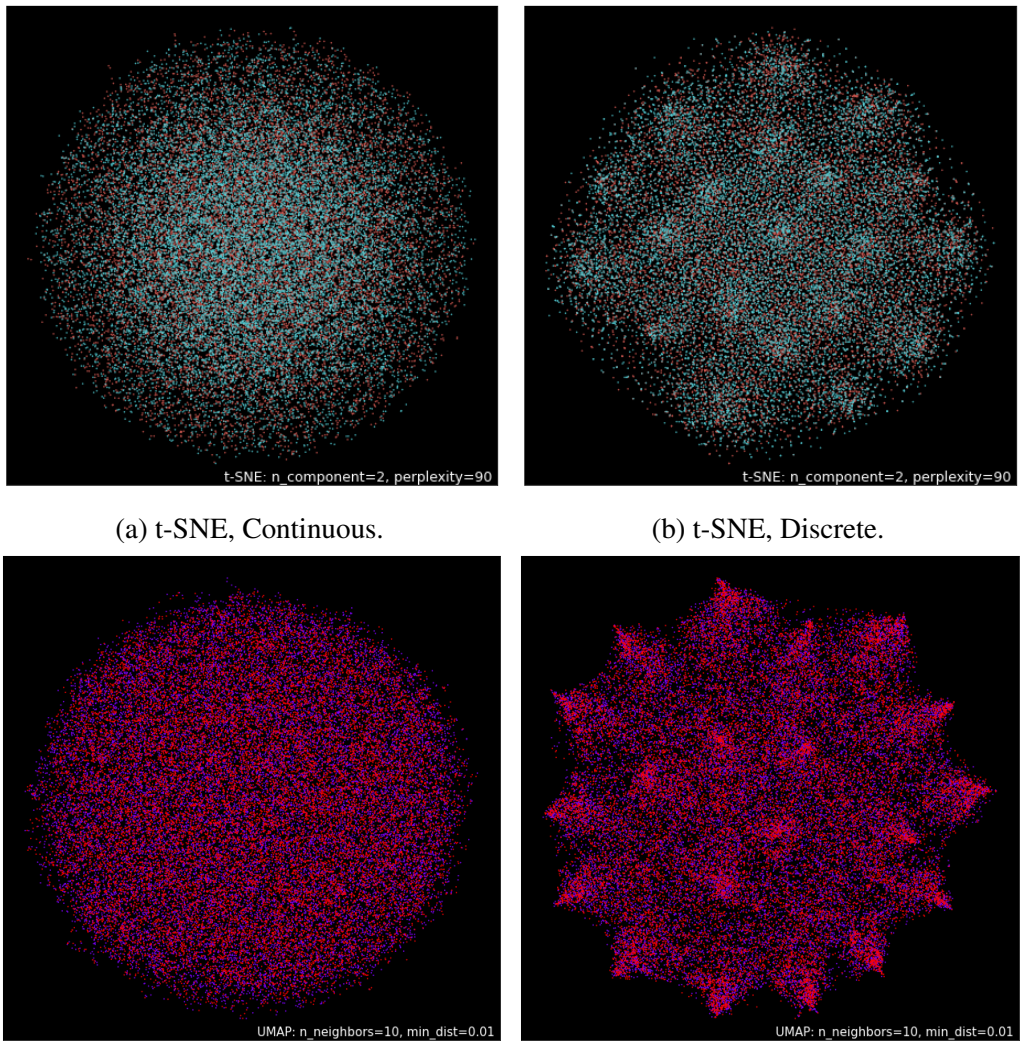
However, if the quantization bits are chosen appropriately, the shift in the distribution should not be significant, and the quantized model should perform similarly to the normal model. In general, it is important to carefully choose the number of bits used for quantization, as this affects the trade-off between the accuracy of the model and the amount of computation required. By understanding the potential impact of quantization on the distribution of layer weights, we can make informed decisions about the trade-off between computation and accuracy in deep learning models.

3.3.5 Visualizing Latent Space

Dimension reduction is a crucial technique in data science for both visualisation and machine learning pre-processing. There are two main categories of dimension reduction algorithms: those that preserve pairwise distance structure and those that favour local distances over global distance. Examples of the former include PCA, MDS, and Sammon mapping, while the latter includes t-SNE, Isomap, LargeVis, Laplacian eigenmaps, and diffusion maps.

Latent space visualization plays a crucial role in comprehending the structure of the data that a Variational Autoencoder has learned. **PCA** (Principal Component Analysis) [126] is a popular technique for dimensionality reduction, however, it can fail to capture the non-linear structure of the data, especially in the case of JointVAE. This is because the JointVAE model is specifically designed to capture both the continuous and discrete aspects of the data, and PCA is not equipped to handle such complex data structures effectively. PCA works by finding a set of linearly uncorrelated principal components that capture the maximum amount of variance in the data, which can result in a good representation of the continuous data but does not effectively capture the discrete aspects.

In contrast, techniques like t-SNE (t-Distributed Stochastic Neighbor Embedding) [127] and UMAP (Uniform Manifold Approximation and Projection) [128, 129] are well-



(a) t-SNE, Continuous.

(b) t-SNE, Discrete.

(c) UMAP, Continuous.

(d) UMAP, Discrete.

Figure 3.7: Representation of the latent space of the JointVAE model for a subset of the test data. Notice in Figure 3.7b and 3.7d the 20 clusters are visible, which correspond to the categories of the discrete latent space.

suited for visualizing the latent space representation of JointVAE models.

- **t-SNE** maps data points into a lower-dimensional space while preserving pairwise similarities. t-SNE is based on the idea that similar data points should be mapped close together in the lower-dimensional space, while dissimilar data points should be mapped far apart. It achieves this by minimizing the divergence between two probability distributions: one that measures pairwise similarities in the high-dimensional space, and another that measures pairwise similarities in the low-dimensional space. The algorithm iteratively adjusts the positions of the data points in the lower-dimensional space until the two distributions match as closely as possible.

- On the other hand, **UMAP** builds upon mathematical foundations related to Laplacian eigenmaps. It addresses the issue of uniform data distributions on manifolds through a combination of Riemannian geometry and category theoretic approaches.

> UMAP is competitive with t-SNE in visualization quality while preserving a more global structure with superior run-time performance. It can also scale to significantly larger data set sizes and has no computational restrictions on the embedding dimensions, making it a viable general-purpose dimension reduction technique for machine learning.

Both t-SNE and UMAP are non-linear dimensionality reduction techniques that preserve the local structure of the data. This crucial aspect can be applied to the latent space of the JointVAE model, where the discrete and continuous latent variables are often interdependent. These algorithms preserve this interdependence in the lower-dimensional embeddings, resulting in a better representation of the latent space compared to PCA. However, UMAP and t-SNE use different approaches to reduce the dimensionality of data and have specific advantages and limitations.

In general, UMAP is considered better than t-SNE when it comes to capturing the structure of the dataset in a lower-dimensional space. This is because UMAP uses a combination of clustering techniques and analysis of local structure in the data to preserve information about the proximity of the original data in a lower dimension. In this way, UMAP can better preserve the global structure of the dataset while reducing its dimensionality. On the other hand, t-SNE is known for its ability to visualize complex structures in a two-dimensional space, but it may not be able to capture the global structure of the dataset. Regarding the JointVAE model, a dimensionality reduction model such as UMAP can be advantageous as it can better capture the structure of the dataset while keeping the dimensionality reduced. This can help to better identify clusters of similar samples in the latent space. Additionally, UMAP is generally faster than t-SNE, making it a good choice for analyzing large amounts of data.

In the resulting plots shown in Figure 3.7, it can be observed, using UMAP and t-SNE techniques, how the latent space of the JointVAE clusters the dataset in proportion to the categories, or patterns in the discrete latent space representation (Figures 3.7b and 3.7d), while smooth gradients or continuous curves are present in the continuous latent space representation (Figures 3.7a and 3.7c). Additionally, the colour of each data point in the plot can indicate its class labels, such as QCD or the top signal in the dataset under analysis.

3.3.6 Interpolation

Interpolation in VAE and JointVAE is a technique used to explore the latent space of a generative model. In interpolation, the idea is to generate new data points that are intermediate between two existing points in the latent space. This is done by computing the linear interpolation between two latent vectors and decoding the resulting intermediate vectors into the data space. By visualizing the interpolated data points, we can gain insight into the structure of the latent space and how it corresponds to the data space. In a JointVAE, the interpolation is performed in the joint latent space of two data domains. This is useful when working with multi-modal data, as it allows us to explore the relationships between the different domains in the latent space. Interpolation is an important tool for exploring generative models, as it provides a way to visualize the smooth transitions between different data points and gain a better understanding of the relationships between the data and the latent space. Additionally, interpolation can also be used to generate new data points, which can be useful for data augmentation and increasing the size of the

training set. Figure 3.8 represents a set of 10 images plotted using the interpolation of the JointVAE latent space. In the code, the encoder part is used to encode a set of images into a latent space, which is a lower-dimensional representation of the images. The code first selects 10 random images from the latent space and calculates the Euclidean distance between each pair of selected images. The maximum distance between the selected images in the latent space is then computed, which represents the maximum dissimilarity between the selected images in the lower-dimensional representation. Next, the code performs a linear interpolation between two randomly selected images in the latent space according to the formula:

$$z_{interp} = z_1 \cdot \alpha + z_2 \cdot (1 - \alpha) \tag{3.6}$$

where z_1 and z_2 are vectors in the latent space that represent two different images and α is a coefficient that varies from 0 to 1. In this case, α is calculated as $\alpha = i \cdot \texttt{interp step}$, where `interp step` is a step that is calculated as $\texttt{interp step} = \texttt{max distance}/9.0$ and `max distance` is the maximum distance between the selected images in the latent space, which in this case is 10.5244. This expression allows us to generate intermediate points in the latent space that represents an interpolation between two images. This means that it generates 10 intermediate points between the two selected images in the latent space. The decoder part of the VAE is then used to decode these intermediate points back into the original image space. Please note that to the human eye, there is no difference between this series of images. This result is likely to be beyond human perception and it is very challenging for our eyes to distinguish differences in some of the generated images.

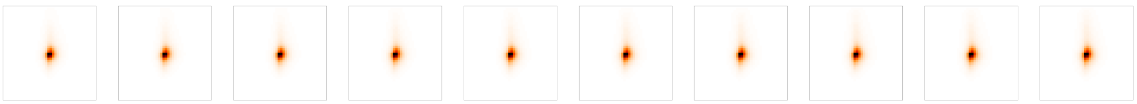


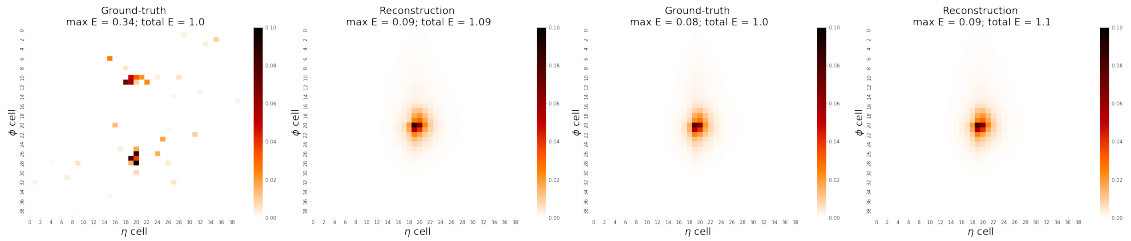
Figure 3.8: The plot shows the 10 decoded images, which represent the interpolation between the two selected images in the original image space. These plots allow us to visualize how changes in the latent space can result in meaningful changes in the original image space, demonstrating the ability of JointVAE models to capture meaningful relationships between images.

3.3.7 Reconstruction Power

Unlike traditional autoencoders that map the input data to a lower-dimensional representation and then reconstruct the original data from that representation, VAEs use a probabilistic approach to reconstruct the input data by sampling the encoded representation from a learned probabilistic distribution. However, this probabilistic approach can sometimes lead to blurry reconstructions and loss of information. To overcome this limitation, the JointVAE architecture disentangles the continuous and discrete latent space data representation, allowing the model to better capture the joint distribution of the input data and its reconstructed output. This joint architecture enables the model to capture both local and global dependencies in the data, leading to improved reconstruction quality. The improved reconstructions from JointVAE can be useful for anomaly detection, which involves identifying data points that deviate from the underlying distribution. By measuring the reconstruction error, the JointVAE can effectively identify these deviations, making

it useful for applications like fraud detection, medical diagnosis, and signal detection in HEP.

AEs and VAEs, such as JointVAE, have a crucial role in the field of HEP as they are used to detect new physics signals that may indicate the presence of unknown particles or phenomena. For instance, in our application, we employed JointVAE to differentiate known QCD from Top quarks jets. However, due to the vast amount of data generated by particle accelerators and detectors, it can be challenging to identify these signals against the background noise. Advanced (V)AEs architectures are designed to accurately capture the distribution of the data, which enables them to identify unknown particles or phenomena that could lead to a significant breakthrough in the field.



(a) QCD reconstruction.

(b) QCD reconstruction means value.

(c) Top reconstruction.

(d) Top reconstruction means value.

Figure 3.9: Reconstruction performance of the model at floating point precision.

Figures 3.9 show the ability of the model to recognize and reconstruct the test images from both the QCD and top jets datasets. The model was trained only on QCD images to use it to recognize the background and so to recognize new signal images that deviate from those. What can be noticed from the plots 3.9a and 3.9b is that the JointVAE can reconstruct QCD images with high fidelity, indicating that the model can capture the important features of the data. However, when it comes to the top quark jet images 3.9c and 3.9d the JointVAE struggles to distinguish them from QCD images, resulting in reconstructions that were very similar to those of QCD images. This result has important implications for anomaly detection, as it suggests that the JointVAE may not be able to reliably distinguish between QCD and top quark jet images. However, it also highlights the potential for using the JointVAE as a tool for discovering new physics in the data, as any significant deviation from the expected reconstruction could indicate the presence of a new signal. In this case, such a deviation in the reconstruction of top quark jet images could potentially point to the presence of new physics phenomena related to top quark production.

3.4 Anomaly Detection scores

As it has been discussed in the section 2.2, anomaly detection is a process of identifying instances in a dataset that deviate significantly from the majority of the data. In the context of JointVAE, several metrics can be used to evaluate the performance of the model in detecting anomalies. These metrics include:

- **Energy Difference**: The energy difference between the reconstructed x' and original data x can be calculated as follows:

$$E_{diff} = \| x - x' \|^2 . \tag{3.7}$$

  The smaller the energy difference, the better the quality of the reconstruction.

- **BCE** (Binary Cross-Entropy): The binary cross-entropy loss between the reconstructed (x') and original data (x) can be calculated as follows:

$$BCE = -\frac{1}{N}\sum_i x_i \log(x'_i) + (1 - x_i)\log(1 - x'_i). \tag{3.8}$$

  This loss measures the dissimilarity between the two distributions.

- **DICE Loss**: The DICE loss [130] measures the overlap between the reconstructed (x') and original data (x) and is defined as follows:

$$DICE = \frac{2\sum_i^N x_i x'_i}{\sum_i^N x_i + \sum_i^N x'_i}. \tag{3.9}$$

  This loss is commonly used in AD tasks and can be used as an additional evaluation metric.

- **Total Loss**: The total loss can be calculated as the sum of BCE and DICE losses:

$$TotalLoss = BCE + DICE. \tag{3.10}$$

- **KL Continuous**: The KL divergence between the estimated continuous distribution $q_\phi(z|x)$ and the prior distribution $p_\theta(z)$ can be calculated as follows:

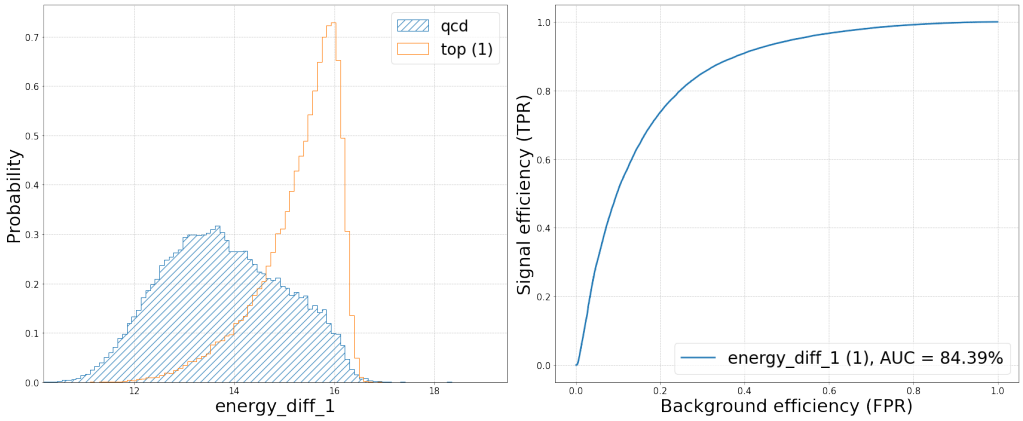
$$D_{KL,cont} = \int_{z\in Z} q_\phi(z|x) \log \frac{q_\phi(z|x)}{p_\theta(z)} dz. \tag{3.11}$$

  This metric is particularly useful for evaluating the performance of the model in capturing the continuous structure of the data.

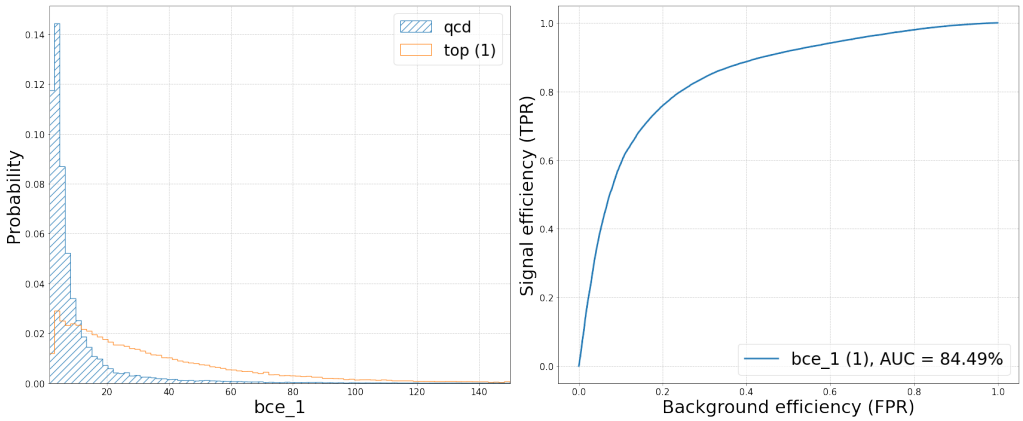
- **KL Discrete**: similarly, the divergence between the estimated discrete distribution $q_\phi(c|x)$ and the prior distribution $p_\theta(c)$ can be calculated as follows:

$$D_{KL,disc} = \sum_{c\in C} q_\phi(c|x) \log \frac{q_\phi(c|x)}{p_\theta(c)}. \tag{3.12}$$

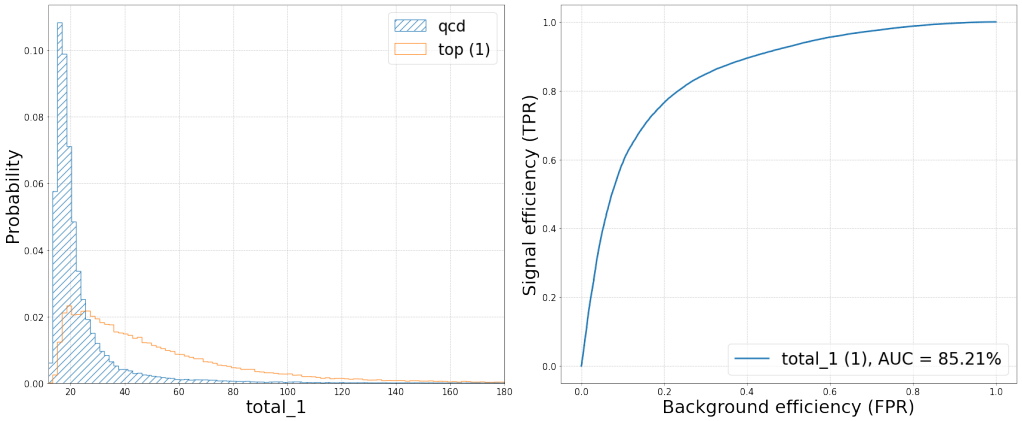
  This metric is particularly useful for evaluating the performance of the model in capturing the discrete structure of the data.



(a) Energy difference score.



(b) BCE score.



(c) Total Loss score.

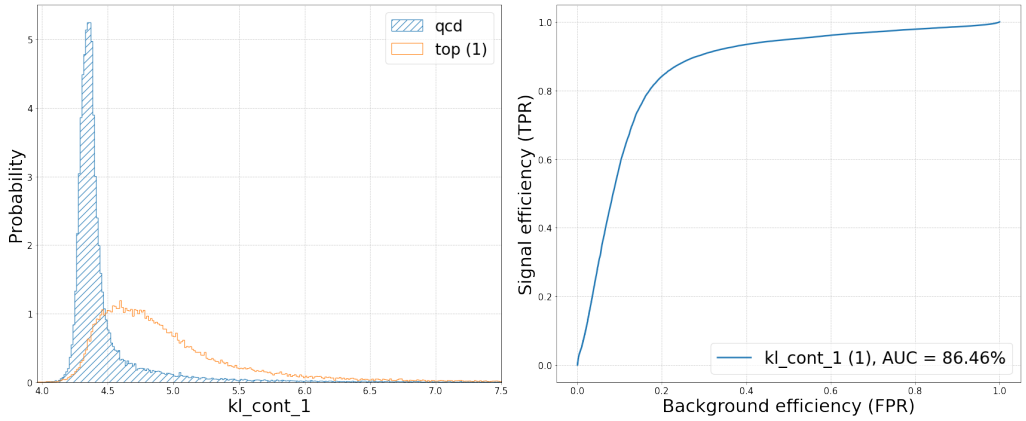
Figure 3.10: AD scores. Notice for the total loss we have an improvement over the base BCE.

- **KL Total**: The total KL divergence is the sum of the KL discrete and KL continuous divergences:

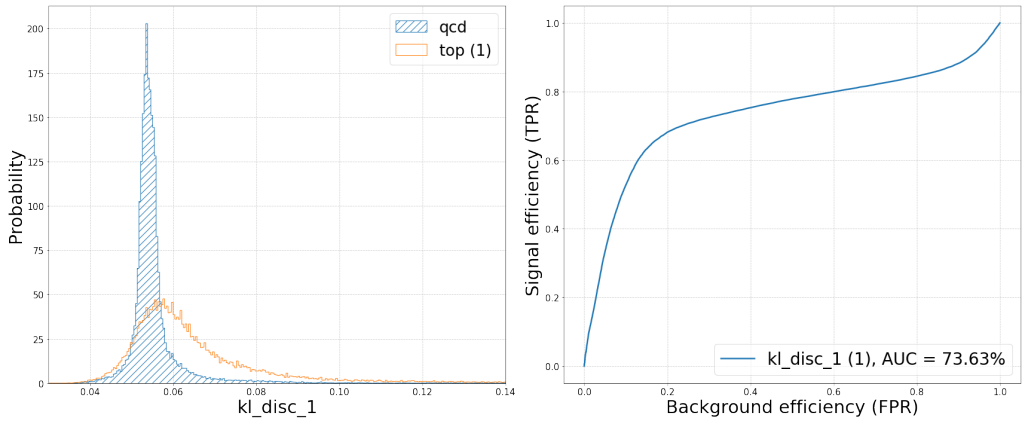
$$D_{KL,tot} = D_{KL,disc} + D_{KL,cont}. \tag{3.13}$$

  This metric is a measure of the overall dissimilarity between the estimated and prior distributions.

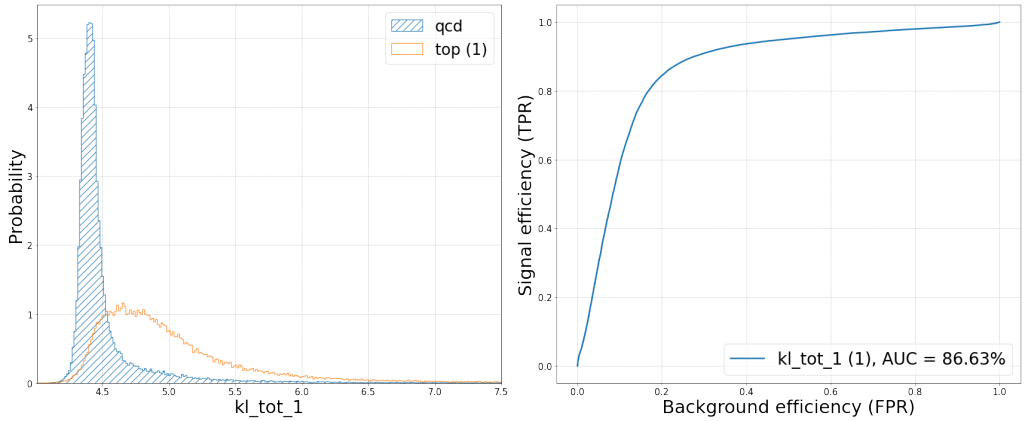
In general, these metrics are used in conjunction to evaluate the performance of the reconstruction and encoding capabilities of the JointVAE model in detecting anomalies.



(a) Continuous KL divergence score



(b) Discrete KL divergence score



(c) Total KL divergence score

Figure 3.11: AD scores. Notice for the total KL divergence there is an improvement over the only continuous KL divergence.

When anomalies are present in the data, they will often result in higher KL divergences, larger energy differences, and larger BCE values. Specifically, these metrics are calculated by comparing the original data to the reconstructed data using both continuous and discrete latent space representations.

In Figure 3.10, we can observe the images for AD results using these metrics. These results show that the JointVAE model can effectively identify instances that deviate significantly from the majority of the data, highlighting its effectiveness for anomaly detection

tasks. Moreover, Figure 3.11 shows the three KL divergences computed in the analysis. The total KL divergence $D_{KL,tot}$, which considers both the continuous and discrete latent spaces, shows an improvement over the single KL divergence. This improvement demonstrates that the inclusion of the discrete latent space in the JointVAE model allows it to capture not only continuous latent space characteristics but also discrete ones. This ability to capture both types of latent space characteristics is not present in the standard VAE model, which can only capture the continuous latent space characteristics, as shown in Figure 3.11a. In conclusion, the addition of the discrete latent space in the JointVAE model has led to an improvement in the performance of the model for anomaly detection tasks. The results presented in Figure 3.11c support this conclusion and show the efficacy of the JointVAE model in capturing both continuous and discrete latent space characteristics.

These metrics can be used to identify instances that deviate significantly from the majority of the data, and thus are candidates for anomaly detection. In Figure 3.10 are shown the images for AD results using the metrics that require both the encoding and the reconstruction capability. In Figure 3.11 we can observe the three KL divergences computed in the analysis mentioned above and it is evident the improvement of the total KL $D_{KL,tot}$ over the single one. This demonstrates the better performance of the JointVAE over the standard VAE which is able only to capture the continuous latent space characteristics, visible in Figure 3.11a. So, adding the discrete latent space has led effectively to an improvement in the performances on the anomaly detection tasks.

Notice also, that is not uncommon for the KL loss in training to be small, as can be seen in the Tab 3.1 in which it has been achieved an order of magnitude of 10^{-4}, even though using KL both continuous, and discrete, and the sum of the score suggests is still effective for anomaly detection. The KL loss is a measure of the discrepancy between the learned latent distribution and a prior distribution. During training, the model learns to approximate the posterior of the latent variables given the observed data. If the model is successful in learning an accurate representation of the data distribution, the KL loss will naturally become small. However, the effectiveness of the KL loss for anomaly detection is not solely determined by the KL loss values during training. Anomaly detection using KL divergence can be based on differences between the learned distribution and the test data, rather than the training data. Additionally, anomaly detection may also rely on the relationship between the latent representations learned by the JointVAE for the background (QCD jet) and the anomalous events. Another possibility is that the JointVAE has learned to identify anomalies by exploiting features that are not captured by the KL loss during the training. For example, the model may have learned to identify anomalies based on higher-order statistics, correlations, or other non-linear relationships between the input variables that are not reflected in the KL loss. Therefore, even if the KL loss is small during training, the KL divergence may still be an effective measure for detecting anomalies in the dataset.

3.5 Compression by Quantization

This section discusses the process of reducing the model footprint to optimize for computational efficiency, specifically when deploying the model on an FPGA. To achieve this, the *encoder* described previously is drastically reduced in size and complexity by reducing both the length and the number of filters, and an additional convolutional layer (conv_fin) was added without calling the activation function to preserve the linearity so far, to help reduce the number of parameters. After this, the final flatten layer was applied,

followed by the output of three dense layers for **mean**, **variance**, and **categorical** values, which correspond to the latent vectors. The table 3.2 presents a summary of the energy consumption, output shape, and model parameters. However, we omitted the instance normalization layers and activation functions in the table since their impact on energy consumption can be negligible.

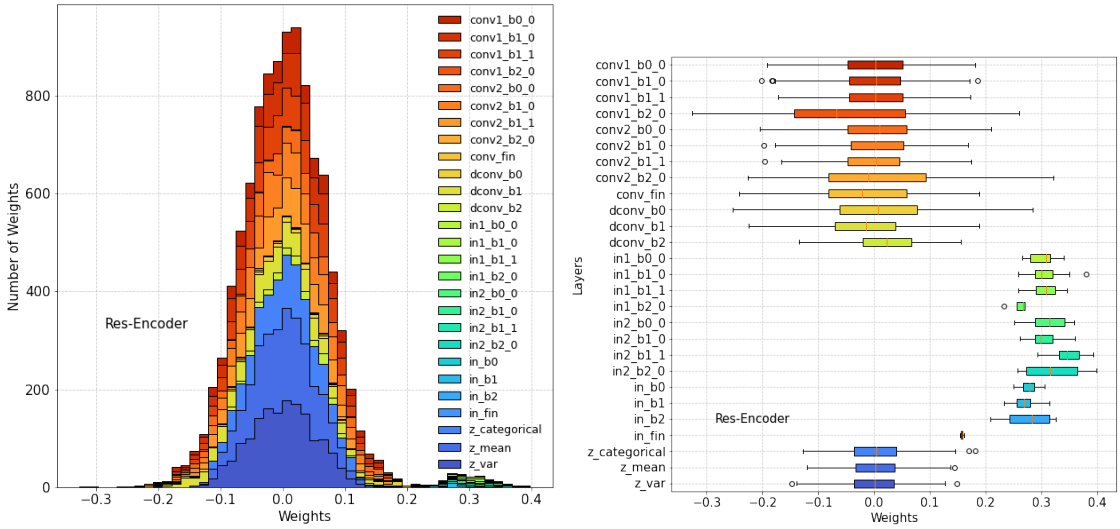
Layer (Type)	Output Shape	Energy [nJ]
dconv_b0 (Conv2D)	(20, 20, 16)	87.9
conv1_b0_0 (Conv2D)	(20, 20, 16)	1382.4
conv2_b0_0 (Conv2D)	(20, 20, 16)	1382.4
dconv_b1 (Conv2D)	(10, 10, 20)	432.0
conv1_b1_0 (Conv2D)	(10, 10, 20)	540.0
conv2_b1_0 (Conv2D)	(10, 10, 20)	540.0
conv1_b1_1 (Conv2D)	(10, 10, 20)	540.0
conv2_b1_1 (Conv2D)	(10, 10, 20)	540.0
dconv_b2 (Conv2D)	(5, 5, 4)	27.0
conv1_b2_0 (Conv2D)	(5, 5, 4)	5.4
conv2_b2_0 (Conv2D)	(5, 5, 4)	5.4
conv_fin (Conv2D)	(5, 5, 2)	2.7
z_categorical (Dense)	(20)	1.5
z_mean (Dense)	(32)	2.4
z_var (Dense)	(32)	2.4

Table 3.2: The Res-Encoder Layers' parameters are displayed. The energy consumption is estimated assuming a 45 nm process [131] using `QTOOLS`.

In addition, the customized layers used in the previous sections of this chapter, which consisted of a `Conv2D` layer with `instance normalization` and an `activation function`, were disassembled and applied separately using native layers defined by the `Keras` and `QKeras` libraries. This is done to facilitate the subsequent conversion with `hls4ml`. To optimize the model for deployment on an FPGA, the numerical precision of the model weights was reduced using a process called *quantization* (see sec. 2.8.2). During training, single- or double-precision floating-point arithmetic is typically used, but when deploying a deep neural network on an FPGA, reduced precision fixed-point arithmetic is often utilized to minimize resource consumption and latency. This process is called Quantization-Aware Training (QAT) and studies have shown that deep neural networks suffer only minor accuracy loss even with binary quantization of weights [132].

The section also describes the model results, after quantization of all its weights, biases, and activation functions are transformed to fixed-point precision before deployment. The selected precision is a new adjustable hyperparameter. With the hls4ml library, users can specify various numerical precisions for different components of the network (referred to as heterogeneous quantization). However, severe PTQ of the activation functions usually leads to greater accuracy reduction than severe PTQ of the weights. By default, hls4ml presumes 16 total bits for every layer, with 6 of them designated to the integer part. We focus on network quantization and explore the use of the QAT approach. By employing QAT, it is possible to maintain high accuracy even with a precision as low as <16, 6>, for some layers. Therefore, QAT is the preferred solution for model quantization before deployment with hls4ml.

To ensure that the network can be synthesized with Vivado HLS and run on an FPGA, the number of filters in the network has to be reduced. Taking in mind that Vivado HLS imposes a constraint on the parameter size for synthesis, which is limited to 1024 per layer. Despite the reduction, the network is still able to detect anomalies with high accuracy. The section also notes that, given the limited space available on the FPGA, the decision is made to only synthesize the encoder model and exclude the decoder. However, during the training phase, the output of the decoder is crucial for properly training the network weights, as it is responsible for reconstructing the input data used to calculate the loss during training. By excluding the decoder during synthesis, the team was limited to using only the KL divergence metric for anomaly detection, which is effective in calculating AD scores. Despite this limitation, the synthesized model was still able to achieve good performance in detecting anomalies. The next sections describe the synthesis process in detail, including the optimization techniques used to achieve the best possible performance, and also discuss the results of the evaluation on the hardware platform, comparing them to the software implementation.



(a) Reduced Encoder weights.

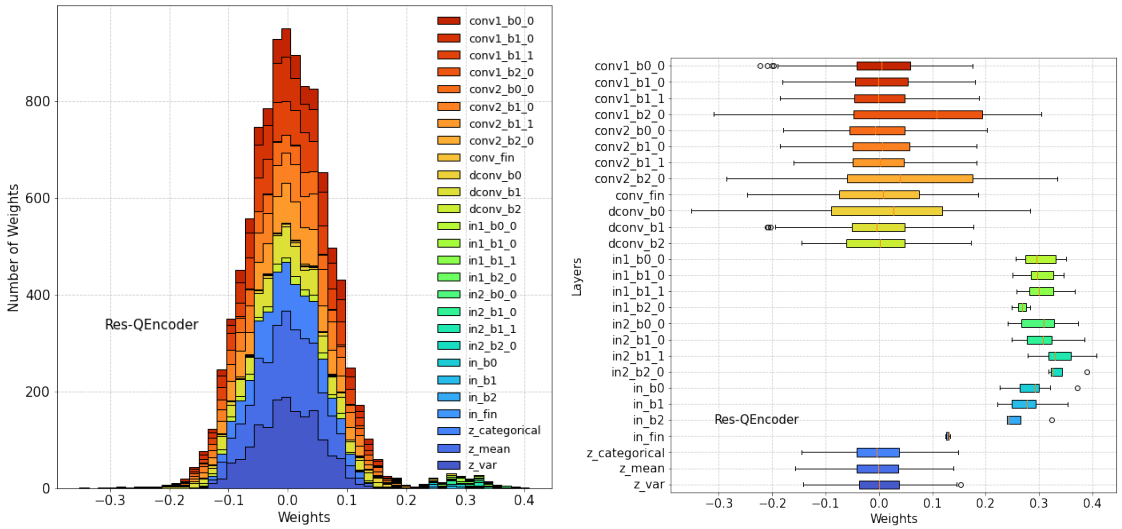
(b) Reduced Encoder whisker plot.

Figure 3.12: Weights distribution for the reduced encoder at floating point precision.

3.5.1 Encoder Performance at Fixed-Point Precision

The description carried on at the beginning of the chapter showed our JointVAE model for anomaly detection performance using the standard Keras library at floating point precision. Now it is discussed the results after quantizing the model with the QAT technique. In particular, it has been imposed a precision of <16,6> to all the convolutional, activation functions and the last final dense layers. We also anticipate that the quantized model will have lower latency and consume fewer resources compared to the floating-point precision model. Furthermore, we expect the quantized model to maintain a high level of accuracy in detecting anomalies. To evaluate the performance of the quantized model, we will use the same dataset and metrics in both the quantized and non-quantized models. Finally, we expect that the synthesized model will have a similar accuracy in detecting anomalies as the software implementation. However, we expect also that the FPGA hardware will be able to process the data at a much faster rate with lower resource consumption due to the

hardware acceleration. We anticipate that these techniques will result in a faster and more resource-efficient implementation of the model on the FPGA.



(a) Reduced Quantized Encoder weights.

(b) Reduced Quantized Encoder whisker.

Figure 3.13: Weights distribution for the reduced and quantized encoder.

As we can see comparing the weights distributions in Figures 3.12a and 3.13a and whisker plots in Figures 3.12b and 3.13b, the quantization bits are chosen appropriately since the shift in the distribution is not significant, and in this case, the quantized model should perform similarly to the normal model.

Model	**AD Scores**	**TPR@FPR** 10^{-1} [%]	**AUC** [%]
Reduced Encoder	KL Discrete	31.46	68.29
	KL Continuous	63.48	84.52
	KL Total	63.63	84.64
Quantized Encoder	KL Discrete	51.67	77.38
	KL Continuous	59.79	80.88
	KL Total	60.03	81.17

Table 3.3: Performance assessment of the floating-point precision and quantized JointVAE, for different AD scores.

Table 3.3 presents the final AD performance for both the reduced encoder model and the quantized version. Although there is a slight decrease in the performance in the total KL AD scores compared to the complete model, as shown in figures 3.11, this can be attributed to the significant reduction in the model size, as explained earlier. Furthermore, the quantization resulted in a reduction in the total KL AD performance, but an increase in the KL discrete performance. Nevertheless, a performance of approximately 81.2% for the AUC of the total KL AD score can still be considered satisfactory for our intended purposes. The ROC curves depict the relationship between the true positive rate (TPR) and the false positive rate (FPR) by adjusting the lower threshold for various anomaly scores. To assess the performance of AD, we measure the area under the ROC curve (AUC) and the TPR at an FPR of 10^{-1} (refer to Table). Looking at the obtained performance we can conclude that three KL divergences can be used as an anomaly metric for the rest of this work.

Model	**Total Energy** [μJ]
Reduced Encoder	5.4957
Quantized Encoder	3.3434

Table 3.4: The energy consumption is estimated assuming a 45 nm process using `QTOOLS`. The quantized model has successfully achieved a reduction in energy consumption.

Table 3.4 indicates that the quantized model has achieved a reduction in total energy consumption by 39.2%. As a result, we anticipate that deploying the JointVAE model for anomaly detection, which has been quantized and implemented on an FPGA, will result in a highly accurate, fast, and efficient implementation. We will present these results in the next section.

3.6 FPGA implementation

In this section of the thesis, we focused on implementing a hardware version of the quantized mode described so far. We first started with a quantized version of the JointVAE model that we aimed to implement in hardware using the HLS4ML tool. However, due to the constraint of the FPGA's size, we had to select only the encoder model that could fit effectively within the available hardware resources. After evaluating various encoder architectures, we chose to use a reduced model size encoder that only included the resulting outputs of the mean μ, variance σ^2, and categorical distribution α. This approach aims to facilitate the computation of discrete, continuous, and overall KL divergence in real-time while conserving energy and ensuring rapid inference for the Anomaly Detection task, which was the intended objective.

However, we encountered a problem when we discovered that the Instance Normalization layer (see section 2.4.2), which is critical for the functioning of our network model, was not implemented in HLS4ML. To address this issue, we proceeded to implement the layer manually and contribute to the open-source repository [104]. To achieve this, we first registered the Instance Normalization layer included in the TensorflowAddons framework by defining a custom layer in the hls4ml's internal representation (IR) model. This custom layer was inherited from the Layer base class and contained the necessary methods for initializing the layer and for computing the forward pass. Next, we generated the corresponding header file in HLS, which contained the HLS implementation of our custom Instance Normalization layer and the effective implementation of the core equation 2.17 used by this layer to achieve the normalization of the sample. The resulting header file contained the HLS implementation of our custom Instance Normalization layer, which is now ready for use in the HLS4ML tool. Finally, we were able to use the `model.converters.keras_to_hls(config)` function from the module `hls4ml`, to specify the target FPGA board, the precision of the inputs and outputs, and the implementation settings.

By manually implementing the layer and contributing to the open-source repository, we not only solved the problem of the missing Instance Normalization layer, but we also helped improve the usability and functionality of the HLS4ML tool for other researchers and practitioners in the field.

3.6.1 Hardware Characteristics

The Alveo U250 [133] data center accelerator card is an ideal candidate hardware platform for implementing the JointVAE model for anomaly detection in the context of the CMS experiment. The card's high-performance FPGA technology provides up to 90× higher performance than CPUs for key workloads, which results in a significant advantage for accelerating the computational-intensive tasks associated with running the JointVAE model. The Alveo U250 card offers a high level of flexibility, allowing users to program the FPGA with their custom hardware accelerators or to use pre-built IP blocks from Xilinx, which makes this hardware ideal for various purposes including machine learning inference, video transcoding, and database search & analytics. These accelerator cards are adaptable to changing acceleration requirements and algorithm standards, capable of accelerating any workload without changing hardware and reducing the overall cost of ownership. Because of this flexibility, the JointVAE model can be optimized for the use case previously described.

The Alveo U250 card also offers high-performance floating-point and integer processing capabilities, which are essential for machine learning and anomaly detection tasks. In particular, the card's peak INT8 performance of up to 33.3 tera operations per second (TOPs) makes it well-suited for implementing the JointVAE model, which requires significant computational resources.

3.6.2 Study of the FPGA implementation feasibility

Results obtained from the implementation feasibility study of the quantized JointVAE encoder model on an FPGA are presented in this subsection. The study focuses on the performance of the model when trained on data from the QCD and top jets datasets. During the conversion of the QKeras model to an HLS project, the model's quantization configuration is passed to hls4ml and enforced on the FPGA firmware. The firmware is designed to use specific arbitrary precision based on the quantization configuration to ensure that the same precision is maintained during inference. The precision of the model parameters is critical in preserving the accuracy of the JointVAE model. This procedure ensures that the use of specific arbitrary precision in the QKeras model is maintained during the inference [106].

To improve the implementation of the JointVAE encoder model, various strategies such as modifying the reuse factor and testing resource or latency strategies were investigated using hls4ml. However, despite these efforts, the model still proves too complex for low-level implementation. This can be attributed to the computational intensity of the model's multidimensional convolutional layers, which cannot be easily simplified for implementation in hardware algorithms. Nevertheless, the addition of the instance normalization layer could prove useful in increasing the variety of networks translatable with hls4ml, although they may not be as complex as the one tested. The primary challenge with high-dimensional inputs, such as images, remains difficult to address. The neural network architecture model used for synthesis required significant computing resources on the development machine used to compile the firmware. For example, to test the accuracy of the model described in the HLS code, a C-simulation was run and it took 31 hours and 53 minutes to process the ∼50000 test images. This highlights the challenges associated with using such models.

Table 3.5 provides a comprehensive overview of the expected resource of the FPGA needed to implement the different layers in the quantized JointVAE encoder model. To examine the resource utilization of the model in detail, we split the blocks and analyzed the resource usage for each layer. Specifically, for each `dconv` block, we considered a `QConv2D`, an instance normalization and `QActivation`. For instance, `conv2_b1_1` represents the first QConv2D layer, instance normalization layer, and activation block with an input and output shape of (10,10,20) and (5,5,4) respectively. A block similar to the previous one named `conv2_b2`, but ingesting an image with dimensions (5,5,4) and generating an output with the same. Moreover, the final block in the quantized JointVAE encoder model comprises `conv_fin`, which is a QConv2D layer with instance normalization but no activation, and three vectors of QDense, which provide the final output of the encoder and the vector defining the latent space of the JointVAE.

The quantized JointVAE encoder is a complex neural network that requires careful optimization for efficient implementation. To this end, resource utilization metrics play a critical role in evaluating the performance of the model and optimizing its implementation. In particular, the number of DSPs, LUTs, FFs, and BRAM used by each layer in the model provides valuable insights into the resource requirements of the model and helps identify potential performance bottlenecks that need to be addressed. It is worth noting that in the actual model intended to be implemented on the FPGA, the blocks present in 3.5 are present multiple times with different and bigger inputs and outputs. This architectural choice is made for better encoding computation, as described in the chapter and it is possible to look at the layer names presented in Table 3.2 made for a floating-point precision of the model for a comparison of the layer names. However, it is important to note that the first `conv2_b0_0` layer in the table, which takes an input of (20,20,16) and has an output shape of (10,10,20), was impossible to simulate with Vivado HLS due to its large shape in convolutional computation. Assuming that the resource consumption would scale proportionally with the input size, the resource usage of this layer was estimated based on a linear relationship with a factor computed according to:

$$\rho(a,b) = \frac{(h^a_{in}/s) \times (w^a_{in}/s) \times d^a_{in} \times d^a_{out}}{(h^b_{in}/s) \times (w^b_{in}/s) \times d^b_{in} \times d^b_{in}} \tag{3.14}$$

where a and b represent two convolutional layers, h is the height, w is the width and d the depth for both the input (in) and output (out). The s represents the stride of the layer which in our case is fixed at 2. The equation 3.14 is estimated according to the estimate of the computational complexity of the convolutional layer as described in the Reference [134]. However, it should be noted that this estimate is only an approximation and may not precisely reflect the actual resource demands of the layer. As we can notice, for the layer conv2_b0_0 we have LUT consumption of 6189696, which exceeds more than three times the total resource available on the FPGA target, which demonstrates the impossibility of the implementation in this configuration.

Overall, the resource utilization metrics and analyses presented in this thesis demonstrate that the quantized JointVAE encoder model is too large for efficient implementation on the FPGA. Further research and optimization efforts are needed to compress and optimize the model for efficient FPGA implementation.

Layer Name	$h_{in}, w_{in}, d_{in}, d_{out}$	**DSP**(%)	**LUT**(%)	**FF** (%)	**BRAM**(%)
conv2_b0_0	20,20,16,10	*752*(6)	*6189696*(**358**)	*229536*(∼7)	*288*(5)
conv2_b1_1	10,10,20,5	47(∼0)	386856(22)	14346(∼0)	18(∼0)
dconv_b2	5, 5, 4,4	15(∼0)	309544(17)	9352(∼0)	4(∼0)
final block	5, 5, 4,2	851(6)	91763(5)	28074(∼0)	377(7)

Table 3.5: Resource utilization estimates and latency for some of the quantized JointVAE encoder layers. Resources are based on the Vivado estimates from Vivado HLS 2020.1 for a target clock period of 5 ns on the Alveo U250. The second column represents the variables used for each layer to estimate resource usage. The percentage was computed against the total available resource on the FPGA target board.

3.7 Summary and Outlook

Conventional methods for detecting new physics at the scales investigated by the LHC may be insufficient in finding interesting results. This creates a critical gap that could be filled by unsupervised machine learning techniques, which could spot anomalous events that would otherwise go unnoticed. To this end, we implement a JointVAE model, a variant of VAE, that learns jointly continuous and discrete variables for better latent representations, targeted at FPGA hardware architecture. The goal was to determine the best latency and resource consumption without sacrificing model accuracy, optimizing the model for classification between anomalous jets and QCD jets images. By training solely on the QCD background, we compared the reconstruction error and a latent space metric to determine the best anomaly detection score that enhances the separation of the two classes. The high-dimensional data representation is transformed into a compressed lower-dimensional latent distribution during the encoding stage. With this application at the LHC, we could ideally classify the jets and even find anomalies using this latent space representation.

After conducting a thorough search for the best model architecture and exploring the hyperparameter space to find the optimal solution, we analyzed several anomaly detection metrics to further optimize the model for FPGA hardware architecture. We utilized a tool called HLS4ML, which enables the implementation of machine learning models on FPGAs. To reduce the model size and energy consumption, we applied the Quantization-Aware of Training technique to the neural networks, achieving a reduction of almost 40% in energy consumption. This process involves mapping the original floating-point values of the model parameters to a fixed-point representation that can be efficiently implemented on the FPGA. It is important to use fixed-point representation instead of floating-point arithmetic because FPGAs have limited resources, and using floating-point arithmetic would result in high resource usage and longer latencies. However, the FPGA implementation results were more challenging than anticipated, and we had to split the block of layers separately to see the performance and make some predictions on the total energy consumption. After analyzing the results, we concluded that the current implementation of the model is not feasible for the FPGA in its current condition, and more efforts are required to compress and optimize the model for efficient FPGA implementation.

To this end, future work can explore the use of *pruning*, this technique has been shown to improve the performance and reduce the computational complexity of deep neural networks. By removing the weights that have a negligible impact on the network's output, the number of computations required during inference can be significantly reduced,

which can improve the speed and energy efficiency of the model. Another area of future research can be the exploration of *knowledge distillation* for improving the quality of the implementation of the JointVAE model on FPGA architecture. Knowledge distillation is a technique where a large, complex model is used to train a smaller, simpler model to achieve similar performance. This technique can be particularly useful in reducing the model size and energy consumption on FPGA while maintaining high accuracy. By leveraging the knowledge of the large, complex model, the simpler model can learn to generalize better and make more accurate predictions. With the use of knowledge distillation, it may be possible to further compress and optimize the JointVAE model for efficient FPGA implementation, enabling the detection of anomalies in real-time with minimal resource usage.

To bring machine learning applications to constrained computing environments, such as detectors in particle colliders, it is necessary to optimize pre-trained models for hardware implementation. This process is challenging because of the limitations of such environments in terms of latency and size. The proposed workflow streamlines this process, resulting in improved physics data collection quality. This research represents a significant contribution to the fields of machine learning and hardware acceleration. The solution proposed for real-time anomaly detection in large datasets in the context of Anomaly Detection in HEP can enhance the performance of machine learning algorithms in physics beyond the Standard Model (BSM). The proposed approach using a joint variational autoencoder is expected to outperform standard deep learning methods and enable the resolution of previously unsolved challenges.

Conclusions

We propose the use of convolutional neural network joint variational autoencoders (JointVAE) trained on a reference standard model (SM) sample to identify top decay jets, but the same technique can be used for potential beyond standard model (BSM) events at the LHC. This approach can detect recurrent anomalies that may be missed by traditional trigger selection, and it can be applied in general-purpose LHC experiments. Our algorithm has the potential to select datasets enriched with events from challenging BSM models, in a similar approach as demonstrated for top jets samples. Furthermore, the algorithm can be trained directly on data with little performance loss, enhanced robustness against systematic uncertainties, and simplified training and deployment procedures.

We discussed a strategy for detecting potential anomalies, including using a JointVAE for anomaly detection (AD) by projecting the input's representation in the latent space. The final outcome of this application is a list of anomalous scores that experimental collaborations could scrutinize further. The purpose of this application is not to enhance the signal selection efficiency for BSM models but to provide a high-purity sample of potentially interesting events. Repeated patterns in these events could motivate new scenarios beyond the standard model physics and inspire new searches to be performed on future data with traditional supervised approaches. Our proposed approach is general and not sensitive to a particular BSM scenario. While supervised algorithms could give better discrimination capability for a given BSM hypothesis, they would not generalize to other BSM scenarios. The JointVAE, on the other hand, comes with little model dependence and therefore generalizes to unforeseen BSM models. As typical of autoencoders used for AD, our JointVAE model is trained to learn the SM background, but there is no guarantee that the best SM-learning model will be the best anomaly detection algorithm.

During our discussion, we also explored methods to enhance the detection of new physics at the LHC by utilizing the model within the L1T infrastructure of the experiments. We proposed deploying a JointVAE on a field-programmable gate array (FPGA) using the hls4ml library. By quantizing the model, we achieved a low latency and minimal resource utilization. It is important to note that the model quantization configuration plays a significant role in ensuring the accuracy of the inference results. This is a crucial aspect since the FPGA has limited resources, and using floating-point arithmetic would result in high resource usage and longer latencies. Our proposed approach can help extend the physics reach of the current and next stages of the CERN LHC, and the strategy demonstrated for the data stream coming from a CMS L1 selection can be generalized to any other data stream from any L1 selection, allowing scrutiny of the full 100 Hz rate entering the HLT system of ATLAS or CMS.

While our implementation of the JointVAE model is a novel advancement in the field, there are still further upgrades that need to be made. We found that the computational cost of this large network is significant and requires substantial resources, making it difficult to fully implement. One possible solution could be to employ knowledge distillation, whereby a smaller student network learns from the larger one by minimizing the differences between their outputs. Additionally, we observed that in an FPGA environment where multiple algorithms are running, it may not be feasible for the JointVAE to utilize all available resources for optimal performance. Nonetheless, the JointVAE has the potential to be a valuable tool in L1 trigger systems, enabling more efficient and accurate detection of anomalous signals that could signify new physics signatures.

The work presented in this thesis has been accepted as an oral presentation at the forthcoming "International Symposium on Grids & Clouds 2023" (ISGC 2023) in Taipei (Taiwan), 19-31 March 2023, in the conference track on "Physics and Engineering Applications".

Bibliography

[1] Lyndon Evans and Philip Bryant. "LHC machine". In: *Journal of instrumentation* 3.08 (2008), S08001.

[2] *CERN Website*. Accessed on December 31, 2022. URL: https://home.web.cern.ch/.

[3] Y Nie et al. "Numerical simulations of energy deposition caused by 50 MeV-50 TeV proton beams in copper and graphite targets". In: *Physical Review Accelerators and Beams* 20.8 (2017), p. 081001.

[4] *A vacuum as empty as interstellar space*. Accessed on December 31, 2022. URL: https://home.cern/science/engineering/vacuum-empty-interstellar-space.

[5] *Pulling together: Superconducting electromagnets*. Accessed on December 31, 2022. URL: https://home.cern/science/engineering/pulling-together-superconducting-electromagnets.

[6] Serge Dailler. *Cross section of LHC dipole. Dipole LHC: coupe transversale*. Accessed on December 31, 2022. URL: https://cds.cern.ch/record/842530.

[7] *Accelerating: Radiofrequency cavities*. Accessed on December 31, 2022. URL: https://home.cern/science/engineering/accelerating-radiofrequency-cavities.

[8] Burkhard Schmidt. "The High-Luminosity upgrade of the LHC: Physics and Technology Challenges for the Accelerator and the Experiments". In: *Journal of Physics: Conference Series*. Vol. 706. 2. IOP Publishing. 2016, p. 022002.

[9] *ALICE*. Accessed on January 2, 2023. URL: https://home.web.cern.ch/science/experiments/alice.

[10] *ATLAS*. Accessed on January 2, 2023. URL: https://home.web.cern.ch/science/experiments/atlas.

[11] *CMS*. Accessed on January 2, 2023. URL: https://home.web.cern.ch/science/experiments/cms.

[12] *LHCb*. Accessed on January 2, 2023. URL: https://home.web.cern.ch/science/experiments/lhcb.

[13] *TOTEM*. Accessed on January 2, 2023. URL: https://home.web.cern.ch/science/experiments/totem.

[14] *LHCf*. Accessed on January 2, 2023. URL: https://home.web.cern.ch/science/experiments/lhcf.

[15] GL Bayatian et al. *CMS Physics: Technical Design Report Volume 1: Detector Performance and Software*. Tech. rep. CMS-TDR-008-1, 2006.

[16] CMS Collaboration et al. "The CMS experiment at the CERN LHC". In: *JInst* 3 (2008), S08004.

[17] CMS collaboration et al. *The CMS tracker: addendum to the Technical Design Report*. Tech. rep. CERN-LHCC-2000-016, 2000.

[18] Lorenzo Viliani, CMS Collaboration, et al. "CMS tracker performance and readiness for LHC Run II". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 824 (2016), pp. 67–69.

[19] Cms Collaboration et al. "The CMS electromagnetic calorimeter project: technical design report". In: *Technical Design Report CMS. CERN, Geneva* 47 (1997).

[20] Cristina Biino. "The CMS Electromagnetic Calorimeter: overview, lessons learned during Run 1 and future projections". In: *Journal of Physics: Conference Series*. Vol. 587. 1. IOP Publishing. 2015, p. 012001.

[21] CMS collaboration et al. "Energy calibration and resolution of the CMS electromagnetic calorimeter in pp collisions at $\sqrt{s}$= 7 TeV". In: *arXiv preprint arXiv:1306.2016* (2013).

[22] HCAL TDR. "HCAL Technical Design Report". In: ().

[23] Bannaje Sripathi Acharya et al. *The CMS outer hadron calorimeter*. Tech. rep. CERN-CMS-NOTE-2006-127, 2006.

[24] V. Klyukhin. *CMS Magnetic System Model. Encyclopedia.* Accessed on January 6, 2023. URL: https://encyclopedia.pub/entry/10857.

[25] CMS collaboration, CMS Collaboration, et al. "The CMS muon project: technical design report". In: *Technical Design Report CMS. CERN, Geneva* 51 (1997).

[26] Florian Bechtel. "The underlying event in proton-proton collisions". In: (2009).

[27] M Aguilar-Benıtez et al. "Construction and test of the final CMS Barrel Drift Tube Muon Chamber prototype". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 480.2-3 (2002), pp. 658–669.

[28] *MUON DRIFT TUBES*. Accessed on January 6, 2023. URL: https://cms.cern/detector/detecting-muons/muon-drift-tubes.

[29] Jay Hauser, CMS Collaboration, et al. "Cathode strip chambers for the CMS endcap muon system". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 384.1 (1996), pp. 207–210.

[30] M Abbrescia et al. "The RPC system for the CMS experiment at the LHC". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 508.1-2 (2003), pp. 137–141.

[31] Seong-Kwan Park et al. "CMS endcap RPC gas gap production for upgrade". In: *Journal of Instrumentation* 7.11 (2012), P11013.

[32] CMS collaboration et al. “The CMS trigger system”. In: *arXiv preprint arXiv:1609.02366* (2016).

[33] CMS collaboration et al. “CMS technical design report for the level-1 trigger upgrade”. In: *CMS Technical Design Report CERN-LHCC-2013-011. CMS-TDR-12* (2013).

[34] CMS collaboration et al. “Performance of the CMS Level-1 trigger in proton-proton collisions at $\sqrt{s} = 13$ TeV”. In: *arXiv preprint arXiv:2006.10165* (2020).

[35] Attila Rácz and Paris Sphicas. *CMS The TriDAS Project: Technical Design Report, Volume 2: Data Acquisition and High-Level Trigger*. Tech. rep. CMS-TDR-006, 2002.

[36] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[37] Pankaj Mehta et al. “A high-bias, low-variance introduction to machine learning for physicists”. In: *Physics reports* 810 (2019), pp. 1–124.

[38] Yann LeCun et al. “A tutorial on energy-based learning”. In: *Predicting structured data* 1.0 (2006).

[39] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1999.

[40] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *arXiv preprint arXiv:1609.04747* (2016).

[41] Léon Bottou. “Stochastic gradient descent tricks”. In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 421–436.

[42] Sarit Khirirat, Hamid Reza Feyzmahdavian, and Mikael Johansson. “Mini-batch gradient descent: Faster convergence under data sparsity”. In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE. 2017, pp. 2880–2887.

[43] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[44] Agnes Lydia and Sagayaraj Francis. “Adagrad—an optimizer for stochastic gradient descent”. In: *Int. J. Inf. Comput. Sci* 6.5 (2019), pp. 566–568.

[45] Matthew D Zeiler. “Adadelta: an adaptive learning rate method”. In: *arXiv preprint arXiv:1212.5701* (2012).

[46] Thomas Kurbiel and Shahrzad Khaleghian. “Training of deep neural networks based on distance measures using RMSProp”. In: *arXiv preprint arXiv:1708.01911* (2017).

[47] Ilya Loshchilov and Frank Hutter. “Decoupled weight decay regularization”. In: *arXiv preprint arXiv:1711.05101* (2017).

[48] Leo Breiman. “Random forests”. In: *Machine learning* 45 (2001), pp. 5–32.

[49] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *Machine learning* 20 (1995), pp. 273–297.

[50] Danilo Rezende and Shakir Mohamed. “Variational inference with normalizing flows”. In: *International conference on machine learning*. PMLR. 2015, pp. 1530–1538.

[51] Bernhard Schölkopf et al. “Support vector method for novelty detection”. In: *Advances in neural information processing systems* 12 (1999).

[52] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. “Isolation-based anomaly detection”. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 6.1 (2012), pp. 1–39.

[53] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[54] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised representation learning with deep convolutional generative adversarial networks”. In: *arXiv preprint arXiv:1511.06434* (2015).

[55] Florinel-Alin Croitoru et al. “Diffusion models in vision: A survey”. In: *arXiv preprint arXiv:2209.04747* (2022).

[56] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. “Density estimation using real nvp”. In: *arXiv preprint arXiv:1605.08803* (2016).

[57] Durk P Kingma and Prafulla Dhariwal. “Glow: Generative flow with invertible 1x1 convolutions”. In: *Advances in neural information processing systems* 31 (2018).

[58] Wei Ping et al. “Waveflow: A compact flow-based model for raw audio”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 7706–7716.

[59] Barry M Dillon, Radha Mastandrea, and Benjamin Nachman. “Self-supervised anomaly detection for new physics”. In: *Physical Review D* 106.5 (2022), p. 056005.

[60] Ting Chen et al. “A simple framework for contrastive learning of visual representations”. In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607.

[61] Priya Goyal et al. “Self-supervised pretraining of visual features in the wild”. In: *arXiv preprint arXiv:2103.01988* (2021).

[62] Raghavendra Chalapathy and Sanjay Chawla. “Deep learning for anomaly detection: A survey”. In: *arXiv preprint arXiv:1901.03407* (2019).

[63] Chun-Liang Li et al. “Cutpaste: Self-supervised learning for anomaly detection and localization”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 9664–9674.

[64] Barry M Dillon et al. “Symmetries, safety, and self-supervision”. In: *SciPost Physics* 12.6 (2022), p. 188.

[65] Theo Heimel et al. “QCD or What?” In: *SciPost Physics* 6.3 (2019), p. 030.

[66] Thomas Schlegl et al. “Unsupervised anomaly detection with generative adversarial networks to guide marker discovery”. In: *International conference on information processing in medical imaging*. Springer. 2017, pp. 146–157.

[67] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[68] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep sparse rectifier neural networks”. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 315–323.

[69] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. icml*. Vol. 30. 1. Atlanta, Georgia, USA. 2013, p. 3.

[70] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289* (2015).

[71] David E Rumelhart and David Zipser. "Feature discovery by competitive learning". In: *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations*. 1986, pp. 151–193.

[72] Mikhail Belkin et al. "Reconciling modern machine learning practice and the bias-variance trade-off". In: *arXiv preprint arXiv:1812.11118* (2018).

[73] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[74] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[75] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[76] Y-L Chung, S-C Hsu, and Benjamin Nachman. "Disentangling boosted Higgs boson production modes with machine learning". In: *Journal of Instrumentation* 16.07 (2021), P07002.

[77] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. pmlr. 2015, pp. 448–456.

[78] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. "Instance normalization: The missing ingredient for fast stylization". In: *arXiv preprint arXiv:1607.08022* (2016).

[79] Yuxin Wu and Kaiming He. "Group normalization". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 3–19.

[80] Mark A Kramer. "Nonlinear principal component analysis using autoassociative neural networks". In: *AIChE journal* 37.2 (1991), pp. 233–243.

[81] Jeremy Jordan. *Variational autoencoders.* Accessed on January 24, 2023. 2018. URL: https://www.jeremyjordan.me/variational-autoencoders/.

[82] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. "Variational inference: A review for statisticians". In: *Journal of the American statistical Association* 112.518 (2017), pp. 859–877.

[83] Imre Csiszár. "I-divergence geometry of probability distributions and minimization problems". In: *The annals of probability* (1975), pp. 146–158.

[84] Olmo Cerri et al. "Variational autoencoders for new physics mining at the large hadron collider". In: *Journal of High Energy Physics* 2019.5 (2019), pp. 1–29.

[85] Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. "Generating diverse high-fidelity images with vq-vae-2". In: *Advances in neural information processing systems* 32 (2019).

[86] Junting Pan et al. “Video generation from single semantic label map”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3733–3742.

[87] Samuel R Bowman et al. “Generating sentences from a continuous space”. In: *arXiv preprint arXiv:1511.06349* (2015).

[88] Wei-Ning Hsu, Yu Zhang, and James Glass. “Unsupervised learning of disentangled and interpretable representations from sequential data”. In: *Advances in neural information processing systems* 30 (2017).

[89] Gerhard Hessler and Karl-Heinz Baringhaus. “Artificial intelligence in drug design”. In: *Molecules* 23.10 (2018), p. 2520.

[90] Joe G Greener, Lewis Moffat, and David T Jones. “Design of metalloproteins and novel protein folds using variational autoencoders”. In: *Scientific reports* 8.1 (2018), p. 16189.

[91] *TensorFlow - TensorFlow by Google*. Accessed on January 25, 2023. URL: https://www.tensorflow.org/.

[92] *Brain Team - Google Research*. Accessed on January 25, 2023. URL: https://research.google/teams/brain/.

[93] *Cloud Tensor Processing Units (TPUs)*. Accessed on January 25, 2023. URL: https://cloud.google.com/tpu/docs/tpus.

[94] James Bergstra et al. “Theano: Deep learning on gpus with python”. In: *NIPS 2011, BigLearning Workshop, Granada, Spain*. Vol. 3. 0. Citeseer. 2011.

[95] *TensorFlow Model Optimization Toolkit — Pruning API*. Accessed on January 26, 2023. URL: https://blog.tensorflow.org/2019/05/tf-model-optimization-toolkit-pruning-API.html.

[96] Amir Gholami et al. “A survey of quantization methods for efficient neural network inference”. In: *arXiv preprint arXiv:2103.13630* (2021).

[97] Google. *QKeras*. https://github.com/google/qkeras. 2021.

[98] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. “Distilling the knowledge in a neural network”. In: *arXiv preprint arXiv:1503.02531* (2015).

[99] Jianping Gou et al. “Knowledge distillation: A survey”. In: *International Journal of Computer Vision* 129 (2021), pp. 1789–1819.

[100] Andre R Brodtkorb et al. “State-of-the-art in heterogeneous computing”. In: *Scientific Programming* 18.1 (2010), pp. 1–33.

[101] Andrea Bocci et al. “Heterogeneous reconstruction of tracks and primary vertices with the CMS pixel tracker”. In: *Frontiers in big Data* 3 (2020), p. 601728.

[102] INVENT LOGICS. *FPGA Architecture*. Accessed on February 21, 2023. 2014. URL: https://allaboutfpga.com/fpga-architecture/.

[103] Javier Duarte et al. “Fast inference of deep neural networks in FPGAs for particle physics”. In: *Journal of Instrumentation* 13.07 (2018), P07027.

[104] FastML Team. *fastmachinelearning/hls4ml*. 2021. DOI: 10.5281/zenodo.1201549. URL: https://github.com/fastmachinelearning/hls4ml.

[105] Sioni Summers et al. “Fast inference of boosted decision trees in FPGAs for particle physics”. In: *Journal of Instrumentation* 15.05 (2020), P05026.

[106] Claudionor N Coelho Jr et al. “Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors”. In: *Nature Machine Intelligence* 3.8 (2021), pp. 675–686.

[107] Barry Dillon et al. “Better latent spaces for better autoencoders”. In: *SciPost Physics* 11.3 (2021), p. 061.

[108] Ekaterina Govorkova et al. “Autoencoders on field-programmable gate arrays for real-time, unsupervised new physics detection at 40 MHz at the Large Hadron Collider”. In: *Nature Machine Intelligence* 4.2 (2022), pp. 154–161.

[109] Thorben Finke et al. “Autoencoders for unsupervised anomaly detection in high energy physics”. In: *Journal of High Energy Physics* 2021.6 (2021), pp. 1–32.

[110] Emilien Dupont. “Learning disentangled joint continuous and discrete representations”. In: *Advances in Neural Information Processing Systems* 31 (2018).

[111] Thea Aarrestad et al. “Fast convolutional neural networks on FPGAs with hls4ml”. In: *Machine Learning: Science and Technology* 2.4 (2021), p. 045015.

[112] Gregor Kasieczka et al. “The machine learning landscape of top taggers”. In: *SciPost Physics* 7.1 (2019), p. 014.

[113] Torbjörn Sjöstrand et al. “An introduction to PYTHIA 8.2”. In: *Computer physics communications* 191 (2015), pp. 159–177.

[114] J De Favereau et al. “DELPHES 3: a modular framework for fast simulation of a generic collider experiment”. In: *Journal of High Energy Physics* 2014.2 (2014), pp. 1–26.

[115] Matteo Cacciari and Gavin P Salam. “Dispelling the N3 myth for the kt jet-finder”. In: *Physics Letters B* 641.1 (2006), pp. 57–61.

[116] Matteo Cacciari, Gavin P Salam, and Gregory Soyez. “The anti-kt jet clustering algorithm”. In: *Journal of High Energy Physics* 2008.04 (2008), p. 063.

[117] Matteo Cacciari, Gavin P Salam, and Gregory Soyez. “FastJet user manual: (for version 3.0. 2)”. In: *The European Physical Journal C* 72 (2012), pp. 1–54.

[118] Sebastian Macaluso and David Shih. “Pulling out all the tops with computer vision and deep learning”. In: *Journal of High Energy Physics* 2018.10 (2018), pp. 1–27.

[119] Eric Jang, Shixiang Gu, and Ben Poole. “Categorical reparameterization with gumbel-softmax”. In: *arXiv preprint arXiv:1611.01144* (2016).

[120] Irina Higgins et al. “beta-vae: Learning basic visual concepts with a constrained variational framework”. In: *International conference on learning representations*. 2017.

[121] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. “The concrete distribution: A continuous relaxation of discrete random variables”. In: *arXiv preprint arXiv:1611.00712* (2016).

[122] *Google Colaboratory*. https://colab.research.google.com/. Accessed February 20, 2023.

[123] Kaiming He et al. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[124] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

[125] Russell D Larsen. “Box-and-whisker plots”. In: *Journal of Chemical Education* 62.4 (1985), p. 302.

[126] Ian T Jolliffe and Jorge Cadima. “Principal component analysis: a review and recent developments”. In: *Philosophical transactions of the royal society A: Mathematical, Physical and Engineering Sciences* 374.2065 (2016), p. 20150202.

[127] Laurens Van der Maaten and Geoffrey Hinton. “Visualizing data using t-SNE.” In: *Journal of machine learning research* 9.11 (2008).

[128] Leland McInnes et al. “UMAP: Uniform Manifold Approximation and Projection”. In: *The Journal of Open Source Software* 3.29 (2018), p. 861.

[129] L. McInnes, J. Healy, and J. Melville. “UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction”. In: *ArXiv e-prints* (Feb. 2018). arXiv: 1802.03426 [stat.ML].

[130] Ruoxi Deng et al. “Learning to predict crisp boundaries”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 562–578.

[131] Mark Horowitz. “Computing’s energy problem (and what we can do about it). In 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)”. In: *IEEE, feb*. 2014.

[132] Itay Hubara et al. “Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations.(2016)”. In: *arXiv preprint arXiv:1609.07061* (2016).

[133] Xilinx. *AMD Xilinx Website*. Accessed on March 7, 2023. URL: https://www.xilinx.com/products/boards-and-kits/alveo/u250.html.

[134] Mark Sandler et al. “Mobilenetv2: Inverted residuals and linear bottlenecks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520.