Master's thesis

Master's Programme in Data Science

Jet Energy Corrections with Graph Neural Network Regression

Daniel Holmberg

February 28, 2022

Supervisor: Prof. Teemu Roos

Examiner: Dr. Henning Kirschenmann

UNIVERSITY OF HELSINKI

FACULTY OF SCIENCE

P. O. Box 68 (Pehr Kalms gata 5)
00014 University of Helsinki

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty Faculty of Science	Koulutusohjelma — Utbildningsprogram — Degree programme Master's Programme in Data Science

Tekijä — Författare — Author

Daniel Holmberg

Työn nimi — Arbetets titel — Title

Jet Energy Corrections with Graph Neural Network Regression

Työn laji — Arbetets art — Level Master's thesis	Aika — Datum — Month and year February 28, 2022	Sivumäärä — Sidantal — Number of pages 70

Tiivistelmä — Referat — Abstract

The LHC particle accelerator at CERN probes the elementary building blocks of matter by colliding protons at a center-of-mass energy of $\sqrt{s} = 13$ TeV. Collimated sprays of particles arise when quarks and gluons are produced at high energies, that are reconstructed from measured data and clustered together into jets. Accurate measurements of the energy of jets are paramount for sensitive particle physics analyses at the CMS experiment. Jet energy corrections are for that reason used to map measurements towards Monte Carlo simulated truth values, which are independent of detector response.

The aim of this thesis is to improve upon the standard jet energy corrections by utilizing deep learning. Recent advancements on learning from point clouds in the machine learning community have been adopted in particle physics studies to improve jet flavor classification accuracy. This includes representing jet constituents as an unordered set, or a so called "particle cloud". Two high performant models suitable for such data are the set-based Particle Flow Network and the graph-based ParticleNet. A natural next step in the advancement of jet energy corrections is to adopt a similar methodology, only changing the problem statement from classification to regression.

The deep learning models developed in this work provide energy corrections that are generically applicable to differently flavored jets. Their performance is presented in the form of jet energy response resolution and reduction in flavor dependence. The models achieve state of the art performance for both metrics, significantly surpassing the standard corrections benchmark.

Avainsanat — Nyckelord — Keywords

Jets, Particle Physics, CMS, Geometric Deep Learning, GNN, Regression

Säilytyspaikka — Förvaringsställe — Where deposited

Kumpula Campus Library

Muita tietoja — Övriga uppgifter — Additional information

Acknowledgements

The research topic for this thesis was crafted up together with Associate Prof. Mikko Voutilainen and Dr. Henning Kirschenmann last fall. I would like to thank Mikko for providing me with employment at Helsinki Institute of Physics and enrolling me to the CMS Data Analysis School hosted by Fermilab. It allowed me to pick up necessary background information on experimental particle physics and gave me the head start I needed for this endeavor. I want to express my sincerest thanks to Henning for guiding me through this project. He is more knowledgeable on jet physics than one could possible imagine. Thanks also to Prof. Teemu Roos for giving useful feedback on my thesis, especially on sections related to machine learning.

The latter half of my master's work took an unexpected turn when I found myself at CERN in the cloud infrastructure group. During my time here I have had the chance to interact with many open minded and clever people from all corners of the world. My IT supervisor Ricardo Rocha and my physics advisor Dr. Clemens Lange have provided me with interesting ideas and perspectives for my machine learning related tasks here. Thanks also to my colleague Dejan Golubovic who has been a good sport both on and off the football pitch.

However, getting to this point in my studies and career would have been a lot harder without the unending support of family and friends. I want to thank my fellow science students for the good times and the many study sessions we have spent together. I am looking forward to many more years of fun shenanigans. Lastly, I wish to thank my family for always being there for me no matter what.

Geneva, December 2021
Daniel Holmberg

Contents

1. Introduction

The Large Hadron Collider (LHC) is the world's largest particle collider built to experimentally probe the Standard Model of particle physics. Protons are accelerated in opposite directions almost to the speed of light until they are made to collide at certain interaction points, one being at the center of the Compact Muon Solenoid (CMS) detector. The collisions give rise to a cascade of particles, which are recorded by various measurement devices as they traverse through the detector. The signals are combined in an attempt to reconstruct the particles that emerged from the collisions. Monte Carlo simulations are used to aid with the reconstruction process. The components of an event are simulated starting from the hard process that produces collinear parton showers that then undergo hadronization. The detector is also simulated to account for any irregularities in the response. Final state particles are clustered according to the initial state partons they originate from, forming so called jets.

The energy generally differs significantly between the reconstructed jets and simulated counterparts. There are multiple reasons for this: the energy will for example be smeared due to the finite resolution of the detector and energy is also lost to invisible particles such as neutrinos etc. A series of calibration procedures are in place at the CMS experiment to deal with any such discrepancies [1]. However, jet energy corrections can potentially be improved by using machine learning since it has a track record of improving high energy physics analyses [2]. Neural networks especially have proven to be very performant in many LHC applications [3]. Because of the large dataset available for this regression problem, and the impressive gain deep learning has produced for b-jet energy regression [4], neural networks will be used in this work too. The latest developments in the field of deep learning in high energy physics is using graph neural networks (GNNs) where applicable [5]. They enable training on jet constituents in their most generic representation as unordered sets. Graphs of the particles can be created with detector coordinates, giving some initial locality information for the deep learning model to work with. The expressive power of GNNs have proven very useful for jet flavor classification [6, 7], and they are therefore employed to improve jet energy corrections in this study.

The thesis is structured as follows: Chapter 2 gives an overview of some particle physics theory in addition to the experimental setup; the steps in the Monte Carlo event and detector simulation are presented in Chapter 3; the reconstruction and calibration process is discussed in Chapter 4; deep learning theory is introduced from the perspective of geometric deep learning in Chapter 5; the methods and implementation of graph neural network regression for jet energy corrections are explained in Chapter 6; the results of the aforementioned regression are shown in Chapter 7; and lastly, Chapter 8 contains some concluding remarks.

2. Particle physics at CMS

Particle physics is at the heart of our understanding of nature. It describes the fundamental building blocks of the universe, and their interaction. Our current understanding of the phenomena that happen on the smallest scales have successfully been described by the Standard Model, which provides a unified picture of the different particles and the forces between them. To validate particle physics theories collider experiments have been built, with the largest one being the LHC. Particles are accelerated in opposing directions and made to collide at certain interaction points. Detectors such as CMS then measure the outcome of those collisions. In this chapter some of the underlying theory in particle physics will be introduced, and after that the components of the CMS experiment are explained.

2.1 Standard Model

The Standard Model of particle physics describes three of the four fundamental forces of nature: the electromagnetic, weak, and strong interactions. Gravity is the only one not explained by the Standard Model. However, the gravitational force is so weak at the minuscule scale of particles that its effect is negligible. Apart from describing the interactions, the Standard Model is also able to classify all known elementary particles. The model has been remarkably successful with predicting previously unobserved particles and phenomena, and has to this date withstood thorough experimental scrutiny [8].

The Standard Model is a relativistic quantum field theory based on local gauge symmetries. Particles are represented as local excitations of quantum fields. The Standard Model can be written as a gauge group with three continuous symmetry groups, or Lie groups, representing three distinct fundamental forces of nature:

$$SU(3)_c \times SU(2)_L \times SU(1)_\Upsilon. \tag{2.1}$$

The Abelian Lie group $U(1)_\Upsilon$ describes the electromagnetic interactions and the subscript Υ represent the weak hypercharge. The non-Abelian Lie group $SU(2)_L$ describes the (electro)weak interactions with a subscript L meaning that only particles with left-handed chirality interact weakly. Lastly, the non-Abelian Lie group $SU(3)_c$ describes strong interactions where c indicates the color charge. Each Lie group has a corresponding gauge field, and all interactions in the Standard Model are mediated by spin-1 gauge bosons which are quanta of the gauge fields. For the electromagnetic field the photon acts as mediator, the weak interaction has the W^+, W^- and Z bosons as mediators, and the strong force is carried by gluons.

Matter particles are represented as spin-1/2 fermions in the Standard Model. They exist in three generations as seen in Fig. 2.1. Each generation contains two quarks: one electrically positively charged up-type and one negatively charged down-type, as well as two leptons: one negatively charged particle and a corresponding neutral neutrino. All stable matter in the universe is made up of the particles in the first generation, which are the lightest of all fermions. The particles in the remaining two generations are heavier and decay into the lighter ones.

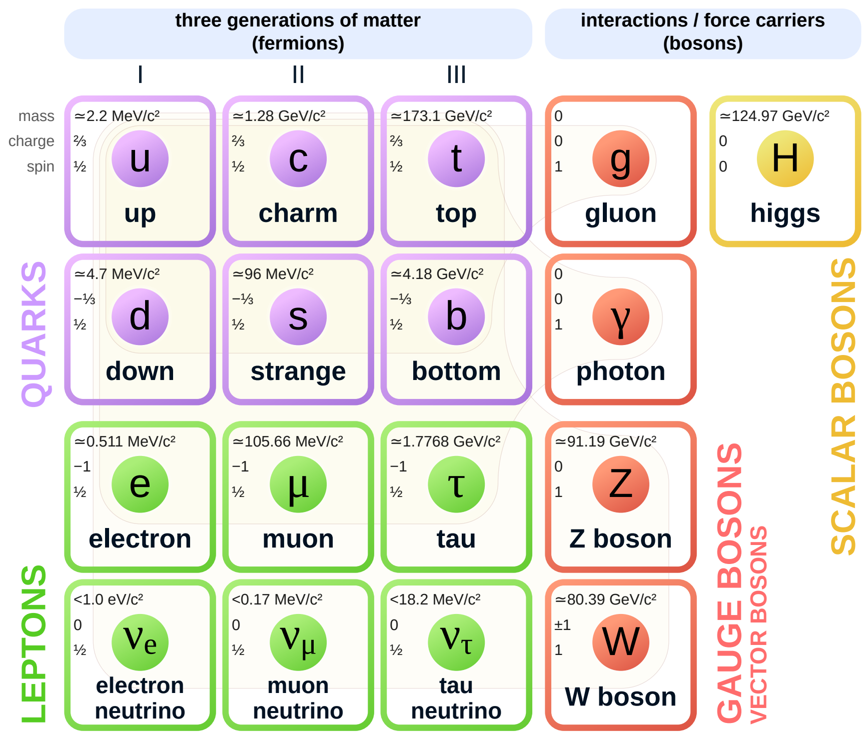

three generations of matter (fermions)
interactions / force carriers (bosons)
I
II
III
mass
charge
spin
≃2.2 MeV/c² ⅔ ½ u up
≃1.28 GeV/c² ⅔ ½ c charm
≃173.1 GeV/c² ⅔ ½ t top
0 0 1 g gluon
≃124.97 GeV/c² 0 0 H higgs
≃4.7 MeV/c² −⅓ ½ d down
≃96 MeV/c² −⅓ ½ s strange
≃4.18 GeV/c² −⅓ ½ b bottom
0 0 1 γ photon
≃0.511 MeV/c² −1 ½ e electron
≃105.66 MeV/c² −1 ½ μ muon
≃1.7768 GeV/c² −1 ½ τ tau
≃91.19 GeV/c² 0 1 Z Z boson
<1.0 eV/c² 0 ½ νe electron neutrino
<0.17 MeV/c² 0 ½ νμ muon neutrino
<18.2 MeV/c² 0 ½ ντ tau neutrino
≃80.39 GeV/c² ±1 1 W W boson
QUARKS
LEPTONS
GAUGE BOSONS
VECTOR BOSONS
SCALAR BOSONS

Figure 2.1: The Standard Model of particle physics. Brown loops indicate which of the five bosons (in red) couple to which of the twelve fermions (in purple and green) [9].

In addition to matter there is also antimatter. Antimatter consists of anti-particles which can be regarded as separate from their regular particle counterparts. However, they are excitations of the same quantum field, and they differ only in that their charges are opposite. A particle and the corresponding anti-particle annihilate when they collide with each other. It is still unknown why there is more matter than antimatter in the universe, but without this asymmetry nothing would exist.

Leptons and quarks interact differently with the fundamental forces. Quarks are affected by the electromagnetic, weak and strong strong force whereas leptons only interact with the electromagnetic and weak forces. The reason for this is that only quarks carry the color charge needed for strong interactions.

All the particles in the Standard Model except the neutrinos obtains their masses from their coupling with the Higgs field. The process is referred to as the *Brout-Englert-Higgs mechanism.* The spin-0 Higgs boson is the visible manifestation of the Higgs field. Where the neutrinos' mass come from is another open question within the Standard Model.

2.2 Feynman diagrams

Interactions between particles in the Standard Model can be represented using *Feynman diagrams.* They are constructed using rules from perturbation theory in quantum mechanics, and give a visualization of transitions between states in particle physics processes. Initial state particles are shown on the left-hand side of a Feynman diagram, and the final state particles are shown to the right. The probability amplitude for a particle to go from one place to another is shown as a line. Lines with arrows represent matter particles whereas squiggly lines represent bosons. For anti-particles the arrows point backwards in time. Particles meet at interaction points shown as vertices, which also have probability amplitudes.

In principle there can be an infinite number of Feynman diagrams for a process by adding an arbitrary amount of intermediary interactions. However, following perturbation theory the simplest, lowest order diagram is the most significant one, and in most cases sufficient for describing a given process. The probability to obtain each final state can be calculated using the total probability amplitude of all the diagrams to some chosen order.

2.3 Quantum chromodynamics

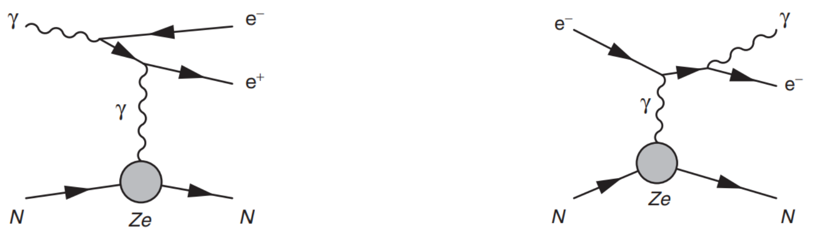
Quantum chromodynamics (QCD) is the theory that describes strong interactions between quarks and gluons, collectively known as partons. QCD has an analog to electric charge called color charge. Borrowing from the RGB color system red, green and blue colors are used in QCD. When combined, these colors create white which is regarded as color neutral in QCD. Combining color and anti-color has the same effect. Baryons are composed of three quarks or antiquarks while mesons are made of a quark and an antiquark. Together they are known as hadrons which are colorless bound states of quarks.

QCD is based on the $SU(3)_c$ gauge symmetry group with eight gauge bosons, or gluons with different mixtures of color charges. Because the symmetry group is non-Abelian, the gluons can interact and exchange colors with themselves. This causes the vacuum to polarize which in turn increases the force linearly with distance greater than about a femtometer. The electromagnetic and weak forces become weaker with increasing distance so in that sense the strong force stands out. It also results in some interesting properties. Firstly, there is *asymptotic freedom* meaning that partons behave as free or weakly bound particles only at very high energies, or equivalently at very small distances. The second phenomenon is called *color confinement* which is that no free color charged parton can exist by itself. If partons in a hadron are separated from each other, the energy will grow until two quark-antiquark pairs are spontaneously produced. So instead of isolating a color charge the initial hadron is turned into a pair of hadrons.

2.4 Particle interactions with matter

To prove or extend theories in particle physics, experiments are set up where particles collide with high energies. New particles emerge from the collisions, but of those only the electron, proton, photon, and the effectively undetectable neutrinos are stable. Unstable particles with lifetimes greater than approximately 10^{-10} s, (e.g. muons or neutrons) can propagate several meters, and can be detected directly by experiments. Some particles are so short lived that they decay before they have traveled far enough from the interaction point to be detected. Only their decay products can thus be detected. The methods for detecting long-lived particles are presented in this section.

The interactions of high energy photons with a medium is dominated by e^+e^- pair production process shown in Fig. 2.2a. When high energy electrons interact with matter they radiate bremsstrahlung photons, which in turn produce e^+e^- pairs. The number of pairs increase exponentially leading to a cascade of electrons, positrons and photons called an *electromagnetic shower*. The average energy $\langle E \rangle$ of the particles created in the shower decreases at a pace proportional to 2^x, where x is the average distance between splits, or *radiation length*. The shower continues to develop until $\langle E \rangle$ falls below a critical energy level, after which electrons and photons lose energy mainly from ionization.



(a) e^+e^- pair production **(b)** Bremsstrahlung

Figure 2.2: Feynman diagrams for high energy photon and electron interactions in the electrostatic field of a nucleus N with charge $+Ze$ [8].

All charged particles can produce bremsstrahlung. However, the rate is inversely proportional to the square of the mass of a particle. Muons are 200 times more massive than electrons, as can be see from Fig. 2.1. Thus, the rate of energy loss to bremsstrahlung is suppressed by a factor of 40 000 for muons compared to electrons. Ionization is for this reason the dominant energy-loss process for muons.

Relativistic hadrons interact strongly with the nuclei of a medium. This primary hard interaction produce new particles, which in turn interact further downstream in the medium. A cascade of particles is formed, called a *hadronic shower*. The distance between interactions in hadronic showers, or *nuclear interaction length*, is significantly longer than the radiation length in electromagnetic showers. Hadronic showers are also much more variable compared to the uniformly behaving electromagnetic showers, because hadronic interactions can produce many different final states. In addition to undergoing strong interactions, charged hadrons lose energy by the ionization process when they traverse a medium.

2.5 Large Hadron Collider

The LHC is the world's most powerful particle collider. It is operated the European Organization for Nuclear Research (CERN), and lies circa 100m underground on the Swiss-French border close to Geneva. The collider resides in a tunnel measuring 27km in circumference. The LHC is designed to collide proton beams with a center-of-mass energy $\sqrt{s}$ of up to 14 TeV [10]. During the last period of data-taking at LHC, Run 2, protons collided with a center-of-mass energy of $\sqrt{s} = 13$ TeV. Protons are made to collide at certain interaction points along the beamline, where large detectors measure what the collisions produce.

Figure 2.3 show the accelerator complex. Hydrogen atoms are ionized at the source, and the resulting protons are accelerated using a linear accelerator up to an energy of 50 MeV. From there they go into the Proton Synchrotron Booster giving them an energy of 1.4 GeV. Next, they are injected into the Proton Synchrotron where they are accelerated to 25GeV. Then, in the Super Proton Synchrotron they attain an energy of 450 GeV, and finally after that the protons are fed into the two beams going in opposite direction in the LHC ring where each of them can have an energy of up to 7 TeV corresponding to a speed of $0.99999999c$, where c is the speed of light.

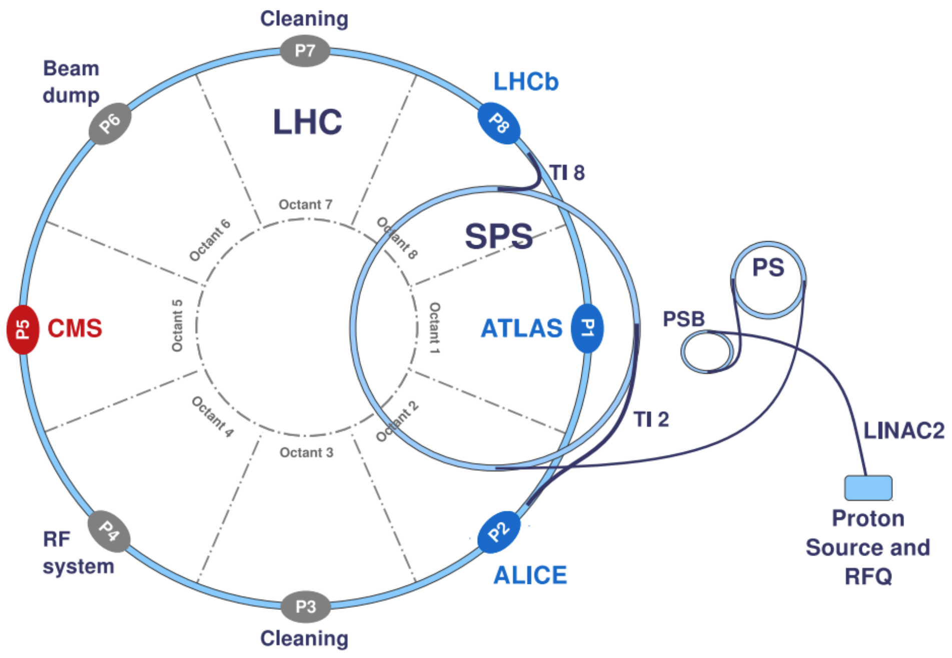


Figure 2.3: Schematic of the LHC complex as seen from above. Protons emerge from the source station, and go through multiple preaccelerators before finally entering the LHC. Collisions take place at four crossing points. Detectors are positioned around these crossing point to take measurements, with the largest being CMS, ATLAS, ALICE and LHCb [11].

The protons in the two LHC pipes are grouped into bunches with a 25 ns time delay, or *bunch spacing*, between them. This is equivalent to a bunch crossing frequency of about 40 MHz. At the interaction points the beams are merged into one leading to the bunches of protons colliding with each other. Multiple interactions happen for every bunch crossing, which is a phenomenon called *pileup*. A higher pileup make it more difficult to determine where the collision products originate from.

To increase the probability of detecting rare particle interactions at LHC the *luminosity* is increased. It's a measure of the number of potential collisions per surface unit over a given period of time. Instantaneous luminosity $\mathcal{L}$ is defined as:

$$\mathcal{L} = f\frac{n_1 n_2}{A} = \frac{1}{\sigma}\frac{\mathrm{d}N}{\mathrm{d}t} \tag{2.2}$$

where f is the bunch crossing frequency, n_1 and n_2 are the number of protons in each colliding bunch respectively, and A is the overlapping cross-sectional area of the bunches. Because A usually is not known very precisely, luminosity often times is expressed in terms of the event rate $\mathrm{d}N/\mathrm{d}t$ and the interaction cross section σ, which describes the likelihood of a process to occur. The LHC is designed for a luminosity of 10^{34} cm^{-2}s^{-1}.

2.6 Compact Muon Solenoid

The CMS is a general purpose particle detector located at the LHC access point 5 as seen in Fig. 2.3. With a weight of 14 000 tonnes, a length of 29 m, and a diameter of 15 m, it is considered to be a *compact* particle detector. One of the distinctive features of CMS, is that it has large gas chambers allowing for precise *muon* measurements. The CMS detector is built around a large superconducting *solenoid* magnet that produces a 3.8 T magnetic field [12]. The field bends electrically charged particles emerging from collisions at the middle of the detector where the interaction point is located.

The residue created by colliding protons in the center of the detector is measured by surrounding subdetectors that cover most directions. A special right-handed coordinate system, shown in Fig. 2.4, is adopted by the CMS experiment. The x-axis points inwards towards the center of the LHC ring, whereas the y-axis points perpendicularly upwards with respect to the horizontal plane of the LHC ring, and lastly the z-axis corresponds to the direction of the beamline. With these coordinates ϕ can be defined as the azimuthal angle measured from the x-axis in the xy-plane, and θ is the polar angle measured from the z-axis.

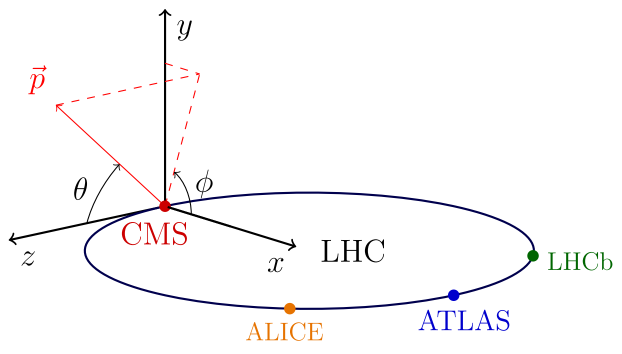


Figure 2.4: A diagram of the CMS coordinate system [13].

The angle of a particle relative to the beam axis is often expressed in terms of the *pseudorapidity* denoted as η, that can be calculated from θ as:

$$\eta = -\ln\left(\tan\frac{\theta}{2}\right). \tag{2.3}$$

Differences in pseudorapidity are invariant under Lorentz boosts along the z axis. This is a useful feature when the colliding partons have different amount of energy because in that case the particles produced come out near one end of the detector. The graph can then be translated to the center-of-mass frame which is easier to analyze. The detector is divided into regions based on pseudorapidity. The *barrel* region $|\eta| < 1.3$, and the *endcap* region $1.3 < |\eta| < 3.0$ are collectively known as the *central* region. The region where $|\eta| > 2.5$ is referred to as the *forward* region.

The CMS detector contains a number of subdetectors responsible for detecting different types of particles. The essential components of the detector is illustrated in Fig. 2.5. The *silicon tracker* sits closest to the interaction point. When charged particles moves through doped silicon wafers in the tracker, the ionization process creates electron-hole pairs. A voltage is applied across the silicon causing the holes to drift in the direction of the electric field to a nearby conductor resulting in a pulse that can be measured. Neutral particles move straight through the tracker.

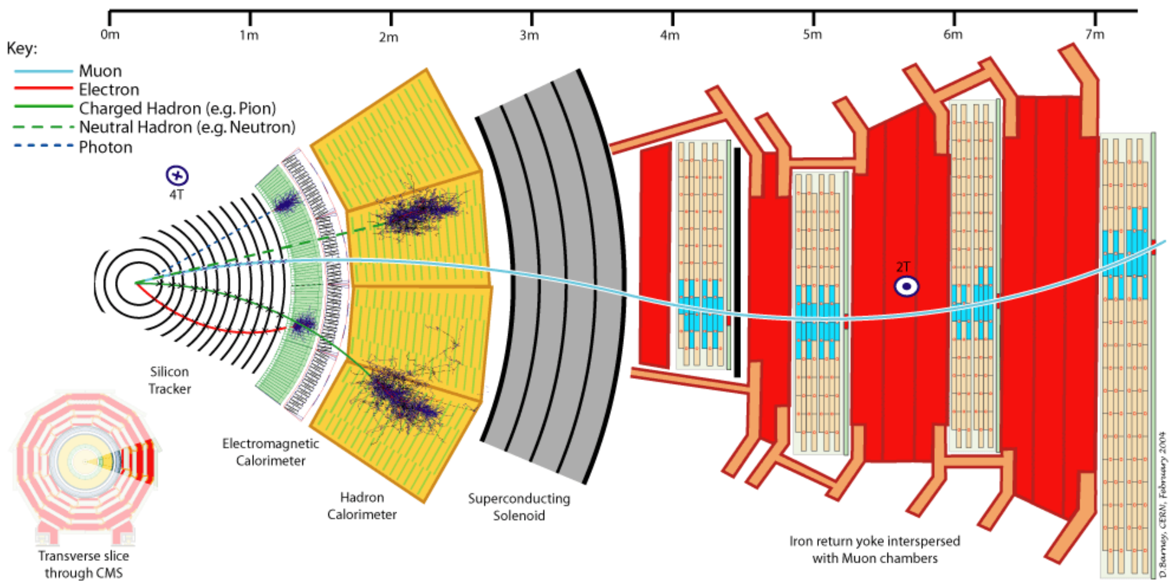

0m
1m
2m
3m
4m
5m
6m
7m
Key:
Muon
Electron
Charged Hadron (e.g. Pion)
Neutral Hadron (e.g. Neutron)
Photon
4T
2T
Silicon Tracker
Electromagnetic Calorimeter
Hadron Calorimeter
Superconducting Solenoid
Iron return yoke interspersed with Muon chambers
Transverse slice through CMS
D.Barney, CERN, February 2004

Figure 2.5: A transverse slice of the CMS detector. Electrons and photons are caught by the ECAL, hadrons are captured in the HCAL, and muons traverse through many layers of steel return yoke, leaving a trail of ionization behind [14].

The second layer in the CMS detector is the *electromagnetic calorimeter* (ECAL). The ECAL is made of lead tungstate which is an optically transparent crystal, and an inorganic scintillator. Scintillation refers to the phenomenon when light of a characteristic spectrum is emitted following the absorption of radiation. Electrons and high-energy photons produce scintillation lights which is then collected and amplified by photon detectors. The amount of light produced is proportional to the energy of the original electron or photon.

The *hadron calorimeter* (HCAL) surrounds ECAL in the CMS detector. The HCAL is responsible for capturing the signal of hadrons. Because of the relatively large distances between nuclear interactions in hadronic showers, the HCAL has to occupy a significant volume of the detector. The HCAL is made up of alternating high-density absorber layers where the showers can develop, and thin scintillator layers for sampling energy deposits. The signals captured by the scintillation layers are summed up to give a measure of the hadronic showers' energy.

The muons produced by the collision lose energy mainly from ionization. They can be distinguished from other charged particles captured by the calorimeters since they only leave trails of ions, whereas all other charged particles interact in other ways as well. Muons are also highly penetrating particles as explained in Section 2.4, meaning that they usually traverse the entire detector. The majority of the CMS detector's volume is occupied by components dedicated to detecting muons, called *muon chambers*. They are interleaved between the steel return yokes in the detector as seen in Fig. 2.5. Muon chambers contain positively charged wires with ionizable gas surrounding them. When muons passes through the chambers they knock electrons off the atoms in the gas, that drift towards the positively charged wires. By registering where along the wire electrons hit, it is possible to calculate the muons' trajectories.

Because collisions happen at a frequency of 40 MHz, it is impossible to store all of the data produced by the CMS detector. Therefore, a *trigger system* is used filter out unwanted data. The initial level-1 trigger (L1) uses very low-level information about the event to decide whether or not to keep it. The data processing for L1 takes approximately 1 μs and is run on field programmable gate arrays (FPGAs). An FPGA is an integrated circuit that can be configured using a hardware description language. It can execute algorithms orders of magnitude faster than a software program. Currently, the L1 trigger outputs events to the high-level trigger (HLT) at a rate of 100 kHz. The HLT is a software based trigger that partially reconstructs events to be able to make more informed decisions of which events to keep. The HLT outputs selected events to offline storage with a frequency of 100 Hz. A very small fraction of all events are saved in the end.

3. Event simulation

Simulations of particle collisions are used to help extract information from the CMS detector. Both the interactions of colliding particles and the detector itself can be simulated because the physical processes that occur in the accelerator and the detectors are known on a theoretical level. Event simulation is based on the Monte Carlo method that uses pseudorandom numbers to simulate event-to-event fluctuations. A large number of events are generated in a way that the probability to produce an event with a given list of final-state particles is approximately that of the real world.

The most prominent Monte Carlo event generators used in high energy physics are Pythia [15] and Herwig [16]. They use a set of parameters for modeling that can vary depending on the *parameter tune* that is used. Examples of such are the CUET [17] and CP [18] parameter tunes. To generate parton-level events there are additional software packages such as MadGraph [19] and Powheg [20]. The event generators give an idealized picture of the measurement in the detector. Inaccuracies in the physical measurement needs to be taken into account, and for that purpose detector simulations are used. Geant4 [21] is used at CMS for simulations of the passage of particles through matter, and is like event simulation also based on Monte Carlo methods.

The various steps used for simulating the collision events are outlined in this chapter. Fig. 3.1 gives overview of the whole process, starting from two protons (red circles) coming from opposite directions, interacting in a process called *hard scatter* depicted as a black blob in the middle. Constituents of the colliding protons interact and produce quarks, leptons, bosons or even hypothetical particles from some new theory. Any new particles with color charge, such as quarks and gluons, radiate virtual gluons which in turn can produce quark-antiquark pairs or emit even more gluons. This leads to the formation of *parton showers* shown as brown in the schematic. The particles give up more and more energy in the iterative parton shower, until eventually triggering the *hadronization* process (yellow ovals). There are also some leftover hadrons from the hard process that are separate from the parton showers creating an *underlying event* seen as a green area in the figure. Detector simulation is added on top of the simulated event to give a realistic picture of the event.

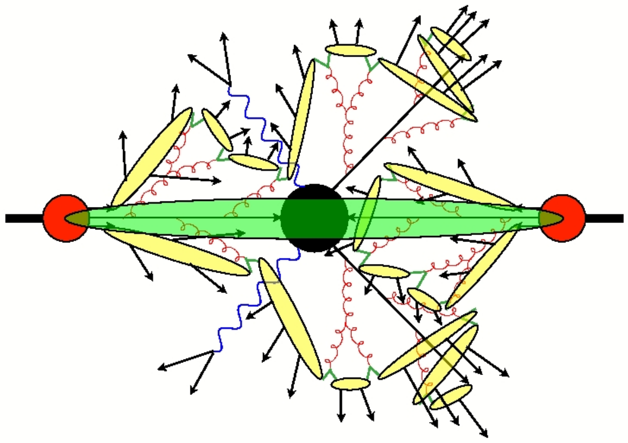


Figure 3.1: An illustration of the simulation steps in Monte Carlo event generator for hadron-hadron collisions [22].

3.1 Hard scatter

The first step of event simulation is to model the hard scatter. Scattering occurs at the center of proton-proton collisions, and it is "hard" in the sense that the momentum transfer is large. This means that the momentum of incoming and outgoing partons have changed significantly due to colliding head on and having high initial kinetic energy. By sampling the *parton distribution function* (PDF) it is possible to get the momenta of interacting constituents. The PDF $f(x, Q^2)$ gives the probability that a parton carries a certain fraction x of the corresponding proton's momentum for an interaction with momentum transfer Q. Because the physics inside protons is non-perturbative the PDFs are determined experimentally at lower energies and in other processes. They are then evolved to higher scales using QCD evolution equations for parton densities. Since PDFs are derived from experiments, they give rise to uncertainty in the simulation process, especially when sampled in the low momentum fraction region.

Hard processes are generated by selecting two partons from the colliding protons as initial particles according to their PDFs. The cross-section, i.e. the probability that a specific process $ij \to kl$ takes place, can be calculated as:

$$\sigma_{ij\to kl} = \int_0^1 \int_0^1 \mathrm{d}x_i \mathrm{d}x_j f_i(x_i, Q^2) f_j(x_j, Q^2) \hat{\sigma}_{ij\to kl} \tag{3.1}$$

where $\hat{\sigma}_{ij\to kl}$ is the parton-level cross section. The latter is a constant derived from integrating the matrix element for the corresponding process. The matrix element can be deduced using rules knowing the Feynman diagram of the interaction. Since the partons are free particles both in the initial and final state, the collision's energy must be high enough for them to achieve asymptotic freedom.

The Monte Carlo method is used by event generators to integrate the cross section by sampling events on random. The precision can be adjusted by varying the amount of terms corresponding to an interaction's Feynman diagram. Currently hard scatter simulations use leading order or next-to-leading order expansion. What has been modeled so far is the very core of the collision, and out flies a couple of partons with high momentum.

3.2 Parton shower

Because the partons emerging from hard scatter are accelerating very fast, they will radiate gluons, the same way accelerating electric charges emit photons. However, unlike the photons which have no electric charge, gluons do have charge in their respective force field, i.e. color charge. Because of that, they can emit further radiation, leading to *parton showers*. The parton splitting can be illustrated using Feynman diagrams, in Fig. 3.2 three possible splits are shown. Color charged partons radiate gluons: $g \to gg$ or $q \to qg$, which can split into a quark-antiquark pair: $q \to q\bar{q}$, or radiate more gluons.

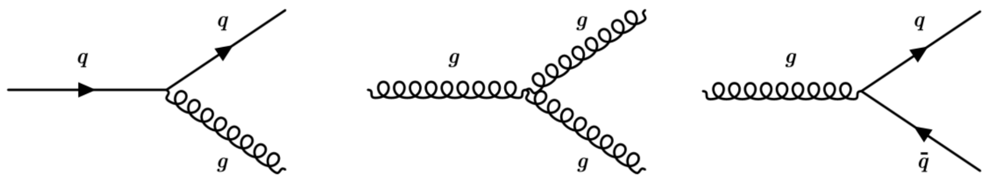


Figure 3.2: Feynman diagrams showing quark and gluon splitting processes that occur in a parton shower [23].

The showers essentially represent higher order corrections to the hard scatter process. They are however unfeasible to calculate in a precise manner. Instead, the dominant contributions associated with colinear parton splitting and soft gluon emission of each order are included in an approximation scheme. Each order's contributions are formulated using probability distributions, leading to a sequence of possible events that depend on the state attained in the previous event. Such a stochastic model is called a Markov process and can simulated using the Monte Carlo method.

Parton showering can be divided into initial state showers, and final state showers depending on when they take place. Initial state showers develop from the incoming partons in the hard scatter, or in other words from the constituents of the colliding protons. Describing the initial shower evolution with PDFs in the forward direction is not really desirable because sampling PDFs have high uncertainty in the low momentum region where the initial state particles would be. So instead the showers are evolved backwards starting from the hard scatter and going backwards in time. Final state showers are the ones emerging from outgoing partons of the hard scatter. For them the initial state momentum is the highest in the beginning of the shower, and thus they are evolved forward in time down until reaching the *infrared scale* $\approx$ 1 GeV, where hadronization takes place and ends the shower.

3.3 Hadronization

When partons emerging from parton showers have low enough energy they combine into hadrons, a direct consequence of color confinement and asymptotic freedom. Perturbative QCD breaks down, requiring a non perturbative model. Such hadronization models treat the partons as a color-connected system. Two prominent models in use are the *string model* used by PYTHIA , and the *cluster model* found in HERWIG .

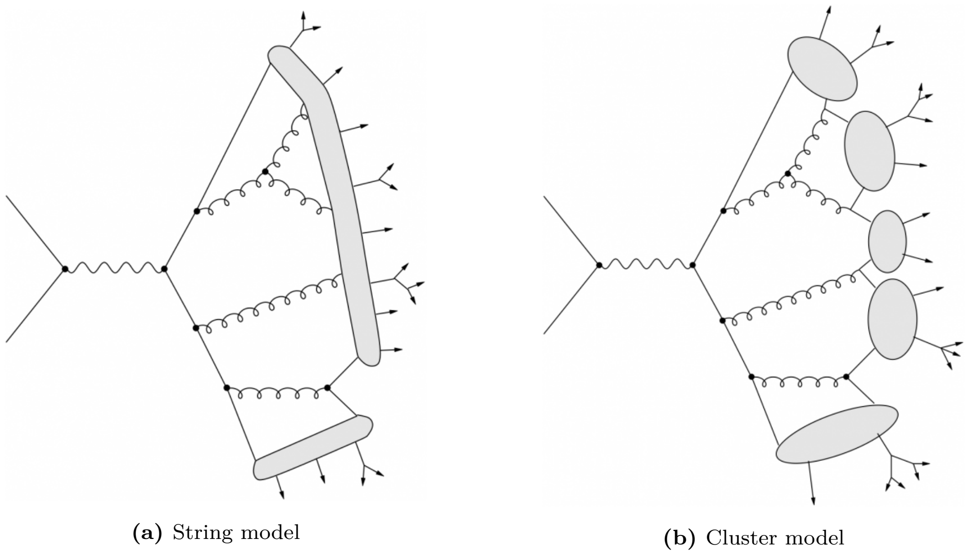

(a) String model (b) Cluster model

Figure 3.3: Illustrations of the string hadronization model on the right, and the cluster hadronization model to the left. In the string model hadrons form a string configuration because of the strong force potential rising linearly between color charged particles. In the cluster model on the other hand quark-antiquark pairs at the end of the parton shower make up clusters from which hadrons are created [22].

The string model, as seen in Fig. 3.3a, is based on the observation that the potential energy of color charged particles rises linearly with their separation. A "color string" so to speak is stretched between them. The reason for this is assumed to be the self-attraction of the gluonic field, causing it to collapse into a string configuration with thickness $\mathcal{O}(1\ \text{fm})$, when the distance between the color charged particles become significantly larger than that.

The cluster model, meanwhile, is based on the concept of *preconfinement*. All gluons are in this model separated into quark-antiquark pairs at the end of the parton shower. After that clusters are formed from colorless combinations of the resulting partons, shown as gray blobs in Fig. 3.3b. Finally, stable hadrons are created from these clusters.

3.4 Underlying event

There are more hadrons produced than what can be ascribed to the showers produced by the hard scatter. The extra activity observed is known as the underlying event, and is believed to arise from those constituents of the incoming protons that are not involved in the hard interaction.

The interactions in the underlying event are soft in the sense that they produce low-p_{T} particles. The interactions consist mostly of diffractive scattering and multiparticle production processes. Because they are soft, perturbative QCD is again not applicable. Instead *multiple-parton interaction models* and *diffraction models* with parameters that can be tuned to experimental results are used by simulations.

3.5 Detector simulation

For the simulated event to match what is observed in reality, the detector itself needs to be included in the simulation chain. Without detector simulation, the simulated event represent purely theoretical particle interactions, which is the kind of *particle-level* information physicists often times want. However, that information is only attainable from measured data by knowing how the detector responds from simulations.

The Geant4 engine used by the CMS experiment can accurately simulate how particles traverse through different materials. A detailed description of all the different components and materials in the CMS detector is provided to the detector simulation software allowing the software to trace particles through the detector step by step using the Monte Carlo method.

Furthermore, Geant4 is able to model various physical interactions with the detector material. Examples of what Geant4 is able to simulate are the effect of electric and magnetic fields, bremsstrahlung, ionization, and interactions between hadrons and nuclei on a wide energy spectrum ranging from MeV-scale elastic scattering of neutrons to TeV-scale hadron showers. The complexity of these effects and the detector setup itself makes detector simulation the most resource intensive steps in the event simulation chain.

4. Event reconstruction

Events measured in the CMS detector are reconstructed with the help of simulated values in an attempt to bring the measurements back to particle-level. Each particle should be individually reconstructed. To achieve this, tracks and energy clusters that lie between the tracker and calorimeter are matched in the Particle Flow algorithm. From the reconstructed data it is possible to group particles together into physics objects known as jets using various jet clustering algorithms. Due to multiple factors such as the detector's nonlinear energy response or its finite energy resolution, the energy of the jets differ from particle-level jet energies. To resolve this issue the jets are calibrated to match simulations in a series of steps laid out at the end of this chapter.

4.1 Particle Flow algorithm

After event and detector simulation, events at CMS are reconstructed using the *Particle Flow* (PF) algorithm [24]. The aim is to map the data measured in the detector back to particle-level. The PF algorithm combines information from all subdetectors to create a global event description containing a complete list of particles (charged hadrons, neutral hadrons, muons, electrons and photons), and their properties (trajectories, momenta and charges). Individual particles are known as PF candidates, which can be used to construct higher-level physics objects. Fig. 4.1 illustrates the workflow of the Particle Flow algorithm.

Tracks are reconstructed using a *combinatorial track finder*. It begins with identifying a track seed, which consists of a few hits that are compatible with some trajectory. Pattern recognition is then used to identify all remaining hits along said trajectory. Lastly, the full track is determined by performing a fit to all hits. The process is repeated multiple times to find the most optimal track.

Electromagnetic showers in the ECAL and hadron showers in the HCAL are wider than one single ECAL crystal or HCAL module respectively. To determine the energies of the

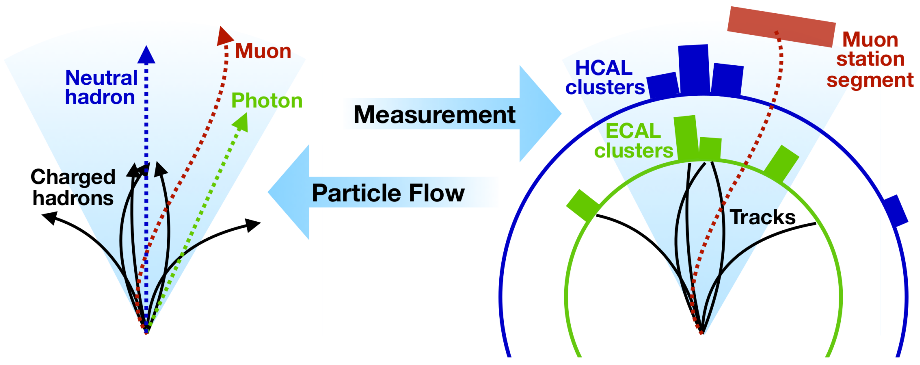


Figure 4.1: Schema for the Particle Flow algorithm. Information from all detector components is used to map the detector signals back to particle-level [25].

particles that initiated the showers the energy deposits in the calorimeter the energy deposits must be clustered. The clustering process begins by selecting calorimeter cells where the energy deposit exceeds some threshold. Neighboring cells are added to the clusters as long as their energy is at least twice as large as a predefined noise level. Lastly, a maximum likelihood fit based on a Gaussian mixture model is used to construct normally distributed clusters of energy deposits that corresponds to the electromagnetic and hadronic showers.

A linking algorithm is used to connect signals which appear to be correlated. The PF algorithm reconstructs particles in the order of the confidence of the linked signals. Once a signal from an identified particle is reconstructed the process is repeated until no more linked signals pass some quality criteria.

Muons propagate through the detector with minimal interaction until detected in the outer rim where the muon chambers are located. This leads to the muons leaving the clearest signals, so they are reconstructed first using information from the tracker and the muon chambers. Electrons are reconstructed from signals in the tracker and ECAL, whereas photons are identified by energy clusters in the ECAL that don't connect to a track. Charged hadrons are reconstructed from tracker, ECAL and HCAL signals. Neutral hadrons on the other hand are a bit trickier to reconstruct since they only leave a substantial energy deposit in the HCAL, and a small signal in the ECAL. However, this has the positive effect that neutral hadrons can be reconstructed separately from charged hadrons even when both of them make energy deposits in the same cell, because the neutral hadrons deposit more energy than expected in the HCAL when compared to the corresponding energy deposit in ECAL.

4.2 Jet clustering

Using a *jet clustering algorithm* physics objects known as *jets* can be created from the hadrons that are observed in the particle detector. The aim of such an algorithm is to cluster a collection of particles observed as a collimated spray in the detector such that they correspond to a parton initiating the particle spray. A jet clustering algorithm has to fulfill two important criteria: 1. it needs to be *infrared safe*, meaning that it is insensitive to soft infrared QCD radiation, and 2. the algorithm should be *colinear safe*, i.e. even if there are collinear splitting of partons, jets are largely clustered the same way.

There are several ways of clustering. However, so called sequential algorithms have proved to work well. They start from a list of particle-level hadrons, and iteratively compare elements of that list, combining them based on some criteria until no more particles can fulfill it. The most commonly used algorithm selects an input particle i at random, and successively compare that particle to other particles, denoted j using distance separation d_{ij}:

$$d_{ij} = \min(k_{t,i}^{2p}, k_{t,j}^{2p})\frac{\Delta_{ij}^2}{R^2} \tag{4.1}$$

where $\Delta_{ij}^2 = (y_i - y_j)^2 + (\phi_i - \phi_j)^2$. The variables k_t, y and ϕ represent transverse momentum, rapidity and azimuthal angle respectively. R is a radius parameter that controls the width of the clustered jet in the (η, ϕ)-plane. If the merging criteria $d_{ij} < k_{t,i}^{2p}$ is met, particle j is added to the jet, and a new j that minimizes d_{ij} is chosen for further comparison. The jet is considered complete when no new particle j fulfills the merging criteria.

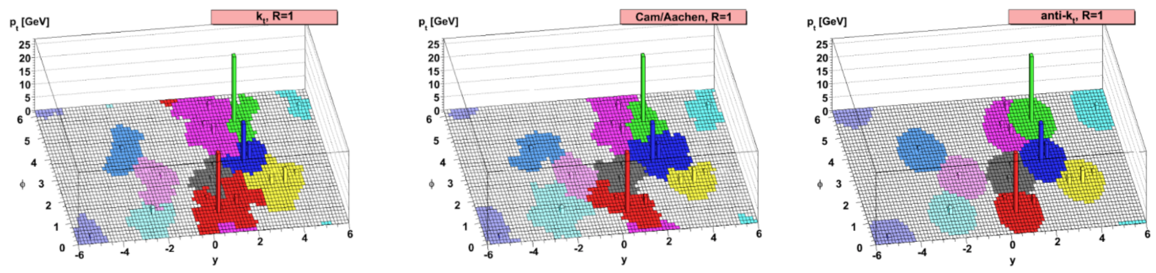


Figure 4.2: Catchments are of jets in a simulated event clustered by the k_t, Cambridge/Aachen and the anti-k_t algorithms [26].

The parameters p controls the relative importance of energy, and algorithms are labeled depending on how it is chosen. The k_t algorithm has $p = 1$, which result in a clustering form the softest inputs to the hardest. If $p = 0$ one gets the Cambridge/Aachen algorithm that has no preference for soft or hard jets in the clustering order. Lastly, the anti-k_t algorithm has $p = -1$, and clusters the hardest inputs first. Interestingly anti-k_t is the most used algorithm of the three, partly because it is both infrared and colinear safe, but also because the jets produced are nicely shaped cones, whereas the other two algorithm produce more irregularly shaped jets, as Fig. 4.2 shows.

In this study jets have been clustered from the PF-candidates using the anti-k_t algorithm. During Run 2 of LHC the distance parameter R was set to 0.4 for light quark jets. Fig. 4.3 summarize how a jet is created from proton-proton collision. A shower of partons is produced by the collision, and color confinement cause the partons to hadronize. The resulting collimated spray of color neutral particles is captured by the detector's calorimeter. The measurements are reconstructed and clustered into a physics object known as a jet.

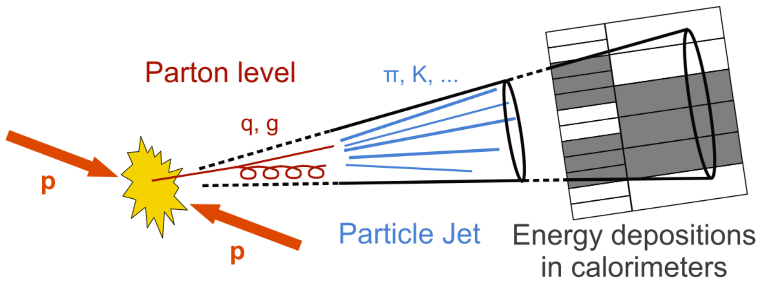


Figure 4.3: Sketch of a proton-proton collision and the resulting collimated spray of particles clustered into a jet [27].

4.3 Jet calibration

The transverse momentum, p_{T}, of the jets that have been clustered from reconstructed PF candidates is different from the true particle-level p_{T}. The reason for this is that the physical detector has a finite energy resolution, and in addition, the mean value for p_{T} will differ due to threshold effects, non-linear detector response, and energy lost to invisible particles such as neutrinos. The aim of *jet energy corrections* (JEC) is to correct the energy, or more specifically the transverse momentum of reconstructed jets, to be as close to particle-level as possible so that the influence of the detector is ignored. JEC consists of multiple steps shown in Fig. 4.4 that each correct for some inaccuracy found in the detector.

4.3.1 Pileup mitigation

The first step of calibrating jets is to mitigate the effect of pileup, both on data and simulated events. Pileup stems from that many unwanted extra soft collision occur in the detector hiding the rare hard interactions. To get more statistics out of the

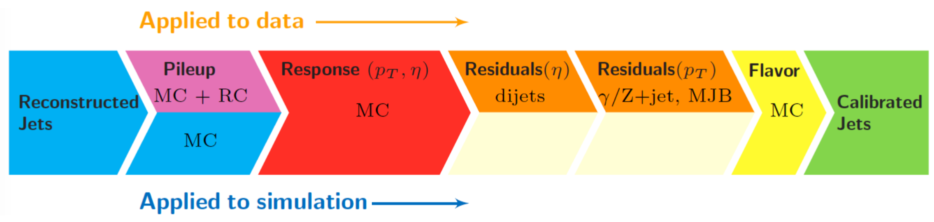


Figure 4.4: The jet energy calibration workflow. Pileup offset corrections are applied to both data and simulated events, likewise with the simulated response corrections that follow. The residual corrections (based on γ + jet, Z + jet and dijet events) are applied only to data. Lastly, the jet flavors are corrected, resulting in calibrated jets [1].

collisions, the luminosity is increased, which means that there are more particles per bunch, decreased bunch spacing and smaller beam size at the collision point. However, increasing the luminosity (as will be done for the upcoming Run 3 of the LHC) leads to even more pileup. The CMS detector can identify the interaction vertex of charged particles, and can as such reject charged particles from pileup vertices. This technique is referred to as charged hadron subtraction (CHS). It identifies about 60% of the energy originating from pileup. To deal with the energy contribution of neutral particles the average energy density of the pileup, ρ, is assumed to be uniformly distributed inside the detector. Then you can perform a simple subtraction to get the pileup corrected transverse momentum $p_{\mathrm{T}}^{\mathrm{corr}}$:

$$p_{\mathrm{T}}^{\mathrm{corr}} = p_{\mathrm{T}}^{\mathrm{raw}} - \rho \cdot A \tag{4.2}$$

where $p_{\mathrm{T}}^{\mathrm{raw}}$ is the raw p_{T} and A is the jet's catchment area. The assumption of uniform distribution of neutral particles is only valid when averaging over multiple collisions, leading to suboptimal reconstruction of single collisions. A newer approach called *pileup per particle identification* tries to address this by assigning weights to all particles individually in a reconstructed event based on their likelihood to be pileup particles. However, the jets in this study used the pileup mitigation strategy outlined above.

4.3.2 Response corrections

The second energy correction targets the response difference caused by the detector itself. Response is defined as $\langle p_{\mathrm{T}}^{\mathrm{reco}} \rangle / \langle p_{\mathrm{T}}^{\mathrm{gen}} \rangle$ where $\langle p_{\mathrm{T}}^{\mathrm{reco}} \rangle$ is the average transverse momentum of the reconstructed jet, and $\langle p_{\mathrm{T}}^{\mathrm{gen}} \rangle$ is the mean of the generator-level jet (both in the same $p_{\mathrm{T}}^{\mathrm{gen}}$ bin). Reconstructed jets are matched to generated jets by requiring the difference in their radius parameter R to be less than 0.2. The correction factor is applied in multiple p_{T} and η bins since the detector is not isotropic and its capability to measure differs depending on a jet's energy. Simulations of the CMS detector, as described in Sec. 3.5, are used to attain an accurate descriptions of the jet response in the detector.

4.3.3 Residual corrections

The difference in jet response that remain after correcting for pileup and simulated response, is dealt with in a series of residual corrections. The law of conservation of momentum is used to estimate the p_{T} of a jet knowing the p_{T} of a recoiling Z boson, photon or another jet with high precision. Residual corrections can be divided into relative and absolute corrections.

Relative residual corrections are determined by extrapolating the precise energy measurements in the barrel region ($|\eta| < 1.3$) to the less precise endcap region ($1.3 < |\eta| < 3.0$) of the detector. Dijet events containing two jets with similar p_{T}, one in the barrel region (tag jet), and one jet in the endcap region (probe jet), are used to accomplish this. A visualization of a dijet event can be seen in Fig. 4.5

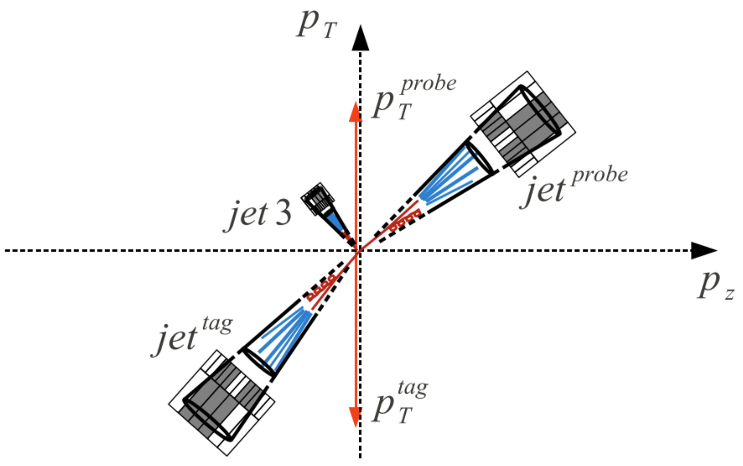


Figure 4.5: An illustration of dijet topology [28].

Absolute residual corrections adjust the jet momenta in the range 30-700 GeV using $Z \to e^+e^-$+jet, $Z \to \mu^+\mu^-$+jet and γ+jet events. Such events contain a Z boson or a photon with precisely measured p_{T} that can be used to estimate the miscalibration and derive a correction for the recoiling jet.

4.3.4 Flavor corrections

Jets are characterized based on the flavor of the quark that initiated the jet in question. Jets receive their flavor by *jet tagging* procedures. Heavy flavored jets are matched to the hardest nearby b or c hadron. Jets with at least one b hadron are labeled as b jets, whereas jets with at least one c hadron and no b hadrons are classified as c jets. Light flavored jets are matched to the hardest nearby u, d, s quark or gluon. Jets that end up without a matching parton are labeled as jets with unknown flavor.

The amount of fragmentation caused by different partons vary. Quark jets, especially those originating from u and d quarks, tend to have the highest response. Owing to their higher color charge, gluons fragments more into softer particles with lower calorimeter response. As a result of heavy hadron decaying into softer particles as well, heavy flavored jets from b and c quarks have a detector response somewhere in between quark and gluon jets.

At CMS the difference in response for different flavors is reduced substantially by replacing the non-linear calorimeter measurement of charged hadron energy with the corresponding track momentum. Despite this, approximately 15% of jet energy that is carried by neutral hadrons is still subject to the calorimeter response non-linearity.

5. Geometric deep learning

Geometry was almost exclusively devoted to Euclidian geometry [29] until the 19th century. At that time mathematicians such as Lobachevsky, Gauss, and Riemann introduced their own versions of new *non-Euclidian* geometries. At the end of the century the different geometries had become separate fields. Mathematicians together with philosophers were debating the validity of them as well as the possibility of one true geometry to rule them all.

In 1872 F. Klein proposed a solution to the diverging state of geometry in the form of the *Erlangen Programme* [30]. The idea is to approach geometry from the perspective of *invariants*, which are properties unchanged under some class of transformations, also called the symmetries of a geometry. Using group theory Klein managed to unify geometry by organizing geometries in a hierarchy of symmetry groups.

The Erlangen Programme had a profound impact on geometry, and the ideas of symmetry also spread, especially to physics where Noether's theorem is a prime example. The theorem states that there are quantities conserved in time (invariants) for every system with a differentiable symmetry property. With given invariants researchers can consider whole classes of hypothetical Lagrangians to describe a physical system. The theorem was absolutely crucial when Yang and Mills in 1954 laid the foundations for the standard model with its underlying symmetry groups.

The current state of deep learning is in a similar place to that of geometry in the 19th century. New model architectures are springing up at a rapid pace which make the field seem very chaotic. Bronstein et. al took inspiration from history, and try to address this issue using symmetry and invariance [31]. By doing so one can classify neural networks into different categories and make them fit into a hierarchical format.

5.1 Learning in high dimensions

Supervised machine learning (ML) refers to learning algorithms that, given a training set of example inputs learn to associate them with a given set of outputs. The training set is commonly referred to as a feature matrix, and can contain a mix of continuous or discrete values. The outputs can also have continuous values, then the prediction task of the ML algorithm is referred to as *regression*. On the other hand, if the outputs are discrete labels, the task is instead *classification*. The name supervised ML stems from that the outputs are provided by a human "supervisor". However, they can be collected automatically as well.

The training data consists of observations $\{(x_1, y_1), \ldots, (x_N,\ y_N)\}$ drawn from a distribution defined over $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X} = \mathbb{R}^d$ is typically a high dimensional input space and $\mathcal{Y}$ is the output space. The labels y are generated by some function $f(x)$ which a supervised machine learning algorithm tries to estimate using a parametrized function class $\mathcal{F} = \{f_{\boldsymbol{\theta} \in \Theta}\}$. Neural networks are an example of this, where the parameters $\boldsymbol{\theta} \in \Theta$ represent the connection weights and potential biases. When fed with an input x, the estimated $\hat{f} \in \mathcal{F}$ provides some output $\hat{y}$ which ideally matches y. The performance of the learned parametrization is evaluated using a loss function $L(y, \hat{y})$, for which popular choices are mean squared error or mean absolute error for regression and binary or categorical cross-entropy for classification.

A neural network is made up of neurons organized in layers. A network with multiple layers is considered a *deep* neural network (DNN) or a multilayer perceptron (MLP). Mathematically, the operation of one network layer can be written as:

$$\hat{f}(\mathbf{x}; \boldsymbol{\theta}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{5.1}$$

where $\mathbf{W}$ is matrix containing all the weight vectors of the individual neurons, and $\mathbf{b}$ is a vector with bias terms for every neuron. The weights and the bias are the learnable parameters of the neural network, that can be collectively written as $\boldsymbol{\theta}$. Some sort of nonlinearity is also needed in order for the neural network to be able to learn something more than a linear mapping of its input. Hence, various *activation functions* are used, such as logistic sigmoid or hyperbolic tangent [32].

Optimization algorithms are used in order to minimize the loss function, and learn the best possible solution to the problem. The most straightforward one is called *gradient descent*. It works by taking the gradient of the loss function with respect to the parameters and use that to update the parameters:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} L(\mathbf{y}, \hat{f}(\mathbf{X}; \boldsymbol{\theta})). \tag{5.2}$$

The factor η is a constant known as learning rate. It is a hyperparameter that adjusts how large steps the model should take in the direction of the gradient at each training step. A learning rate which is too high can result in that a minima won't be found, and a too small learning rate can cause the training to take a very long time to converge, and potentially get stuck in an undesirable local minimum.

A naive direct computation of the gradient is unfeasible since the number of required operations will scale exponentially with respect to the number of edges in the computational graph. However, by using the chain rule and computing the gradient one layer at a time, as well as iterating backwards from the last layer, one can avoid redundant calculations of intermediate terms in the chain rule. In that case the number of operations scale linearly with number of edges in the computational graph. This algorithm, also known as *backpropagation*, was first described in S. Linnainmaa Master's thesis [33], but without the intention of using it for training neural networks.

Multilayer perceptrons (MLP) are, as shown in Fig. 5.1, *universal approximators*. This is very useful since modern day computational capacity allow for the design of rich function classes $\mathcal{F}$ with the capacity to perform approximations on large datasets. However, even if MLPs can approximate target functions arbitrarily well, some form of inductive bias or effective priors that capture the regularities in the learning task are still required for the model to be able to generalize well.

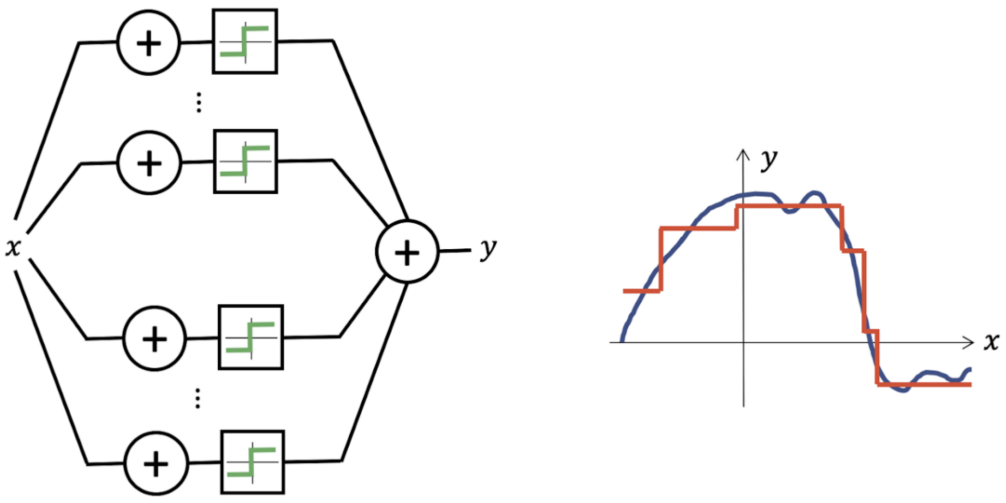


Figure 5.1: A multilayer perceptron with only one hidden layer can represent any combinations of step functions, and can thus approximate a continuous function to an arbitrary precision [31].

For data in low dimensions machine learning algorithms can easily generalize correctly. However, when moving to higher dimensions, learning becomes exceedingly difficult. This phenomenon, as seen in Fig. 5.2, is referred to as the *curse of dimensionality*. The number of possible ways to configure the data is much larger than the number of training examples in a high dimension setting. In order to perform approximations on such a sparse dataset the number data-points needed for analysis grows exponentially. An immediate response to the problem would be to project the samples to lower dimension and perform the learning on that instead. Unfortunately, it is very likely that a substantial amount of fidelity of the inputs is lost by doing such an operation, leading to the need for an alternative source of regularity.

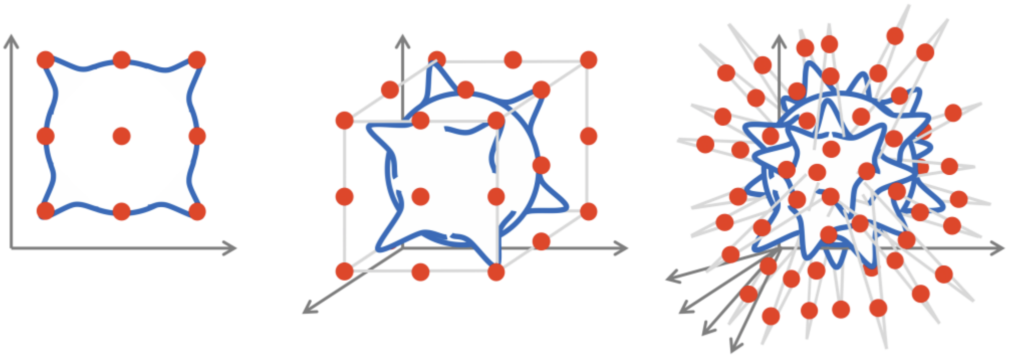


Figure 5.2: Fitting a function in high dimensions is a cursed estimation problem. In order to approximate even a simple class of Lipschitz continuous functions shown here as superposition of Gaussian blobs put in the quadrants of a unit cube, the number of samples grows exponentially with the dimension [31].

5.2 Geometric priors

To overcome the problem of learning in high dimensions, the geometric structure of the domain underlying the input signals, can be used to our advantage. In the geometric deep learning article the principles of *symmetry* and *scale separation* are utilized to combat the curse of dimensionality. Collectively they can be referred to as *geometric priors* for deep learning. A symmetry is a transformation that leaves some system unchanged, whereas scale separation refers to the preservation of important characteristics in the signal when transferring it onto a coarser version of the domain.

The first geometric prior, symmetry, can be explained using an example from computer vision. A classic example is the attempt to classify digits in images as can be seen in Fig. 5.3. The domain in this example is a grid which imposes structure on the class of functions f that the model aims to learn. Functions unaffected by the action of the group $\mathfrak{G}$ are called invariant functions. In image classification a good example of this is shift invariance. No matter where in the image the number three is located the model should still be able to correctly classify it as number three. In some cases the function can have the same input and output structure, for example in image segmentation, where the output is a pixel-wise label mask. In that case the output should be transformed in the same way as the input, which is called an equivariant transformation. In the case of image segmentation this concept would manifest as shift equivariance.

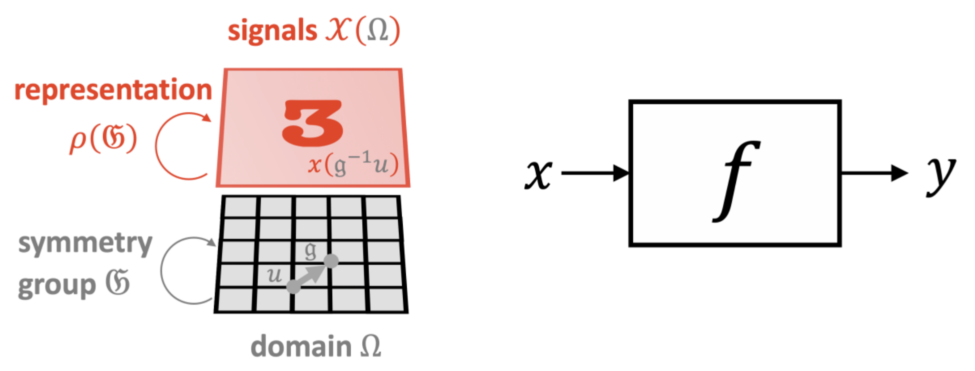


Figure 5.3: The use of geometry in an image classification task. A signal (red square) $\mathcal{X}(\Omega)$ is defined on some domain Ω (gray square), which in this case is a 2-dimensional grid. The structure of the domain Ω is captured by a symmetry group $\mathfrak{G}$, here seen as translations in 2-d space, which acts on the points of the domain $x \in \mathcal{X}(\Omega)$. In the space of signals the group actions on the underlying domain are manifested through what is called the group representation $\rho(\mathfrak{G})$, which here would be the shift operator [31].

In some cases it is possible to construct a multiscale hierarchy of domains by assimilating nearby points. From those domains a hierarchy of signal spaces can be formed that are related to the original signal spaces by a coarse graining operator $P : \mathcal{X}(\Omega) \rightarrow \mathcal{X}(\Omega')$, as illustrated in Fig. 5.4. Coarse-scale functions f' can be applied on the signals on the coarse domain. The function f is said to be locally stable if it can be approximated as the composition of the coarse graining operator P and the coarse scale function f', i.e. $f \approx f' \circ P$. While the original function f might depend on long-range interactions on the domain, in locally stable functions it is possible to separate the interactions across scales, by first focusing localized interactions, and then propagate them towards the coarse scales.

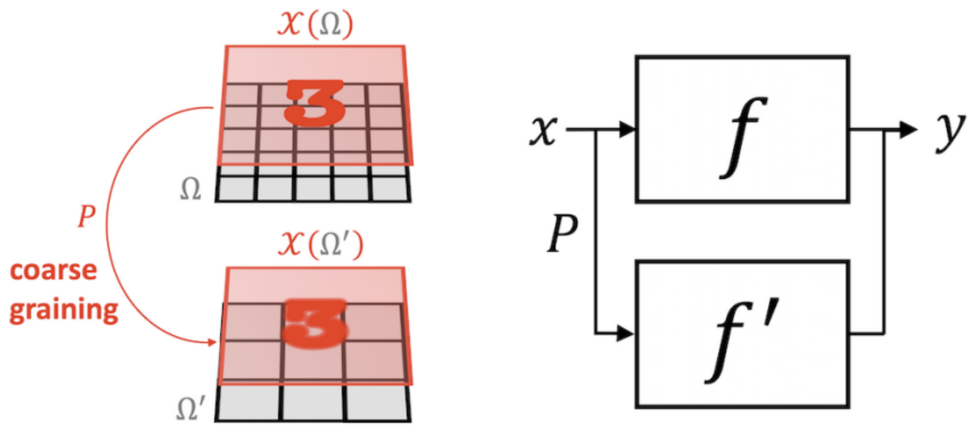


Figure 5.4: Scale separation in an image classification task. The signal $\mathcal{X}(\Omega)$ illustrated as a red three is mapped by P to a signal on a coarse grid Ω' making the coarse signal $\mathcal{X}(\Omega')$ less sharp. The classifier f' acts on the coarse grid signal which should be approximately the same as f operating on the original signal [31].

5.3 Geometric blueprint for deep learning

The two principles of symmetry and scale separation yields a general blueprint of geometric deep learning that can be recognized in popular deep neural network architectures today. An example from computer vision was used to explain the geometric priors, however other prime examples are the rotational invariance (symmetry) used when training on molecular graphs in chemistry, or local pooling (scale separation) when training graph neural networks in general. Using the geometric priors it is possible to derive three key building blocks (Fig. 5.5), that are inherently connected to the geometry of the input domain, and the underlying symmetry group.

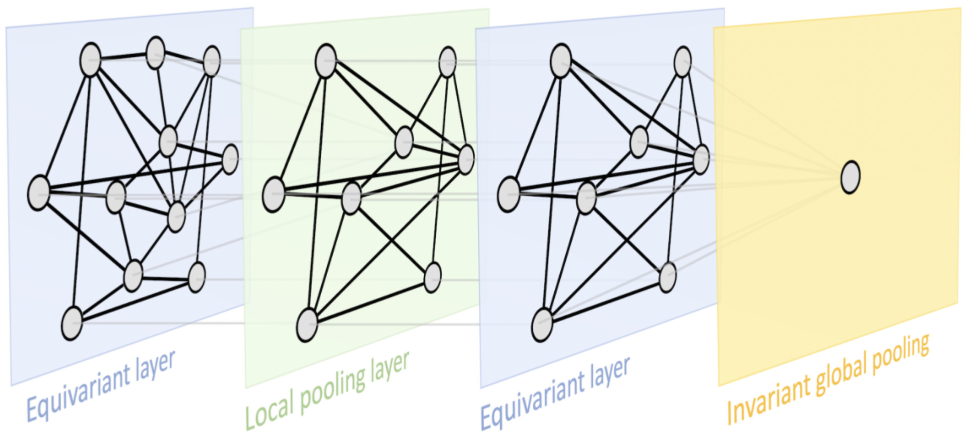


Figure 5.5: A sequence of layers in neural network following the principles of geometric deep learning. Permutation equivariant layers are key to learning on individual nodes. There is a local pooling layer between them that coarses the graph. Lastly, the network has a permutation-invariant global pooling layer as readout layer that maps to the target [31].

First, there is the linear $\mathfrak{G}$-equivariant layer that computes node-wise features:

$$B : \mathcal{X}(\Omega) \rightarrow \mathcal{X}(\Omega'), \text{ satisfying } B(\mathfrak{g}.x) = \mathfrak{g}.B(x) \text{ for all } \mathfrak{g} \in \mathfrak{G} \text{ and } x \in \mathcal{X}(\Omega) \tag{5.3}$$

The second building block is the coarsening operator, also known as local pooling, that can help with learning performance by creating a hierarchy of domains:

$$P : \mathcal{X}(\Omega) \rightarrow \mathcal{X}(\Omega'), \text{ such that } \Omega' \subseteq \Omega \tag{5.4}$$

Lastly, the $\mathfrak{G}$-invariant layer, or global pooling:

$$A : \mathcal{X}(\Omega) \rightarrow \mathcal{Y}, \text{ satisfying } A(\mathfrak{g}.x) = A(x) \text{ for all } \mathfrak{g} \in \mathfrak{G} \text{ and } x \in \mathcal{X}(\Omega) \tag{5.5}$$

may be used to aggregate the output to a global setting if the goal is to make predictions over the domain as a whole. In addition to these three blocks, nonlinearities $\sigma(x)$ are needed to compute nontrivial problems. They come in the form of activation functions in deep learning. With all of these tools at our disposal it is finally possible to construct the desired $\mathfrak{G}$-invariant functions $f : \mathcal{X}(\Omega) \to \mathcal{Y}$ of the form:

$$f = A \circ \sigma_N \circ B_N \circ P_{N-1} \circ \ldots \circ P_1 \circ \sigma_1 \circ B_1 \tag{5.6}$$

where the blocks are chained in such a way that the output space of one block matches the input space of the next one. Different blocks may have different activation functions, and may also exploit different symmetry groups.

The blueprint for learning presented here is at the right level in terms of generality to be applied in a wide range of domains used in current deep learning, which are mainly the “5Gs”: grids, groups, graphs, geodesics and gauges. In table 5.1 various neural network architectures are presented with their respective domain and symmetry group. The domains of interest in this thesis are sets and graphs, so a focus is put on solely those in the next section.

Architecture	**Domain Ω**	**Symmetry group $\mathfrak{G}$**
CNN	Grid	Translation
Spherical CNN	Sphere	Rotation
Intrinsic / Mesh CNN	Manifold	Isometry / Gauge symmetry
GNN	Graph	Permutation
Deep Sets	Set	Permutation
Transformer	Complete graph	Permutation
LSTM	1D Grid	Time warping

Table 5.1: Different deep learning architectures are used on data with different underlying domains. In addition, they use symmetry groups tailored to the domain such that the transformations are invariant [31].

5.4 Learning on sets and graphs

Graphs can often be use to represent different types of data. This thesis deals with particle collision data that naturally comes in the form of unordered sets, and by using detector coordinates for example, one can also form a graph of the data. Learning on graphs is a particular setting of the geometric blueprint for learning with the domain Ω as a graph $\mathcal{G}$. The symmetry is given by the permutation group $\mathfrak{G} = \sum_n$ with all the possible orderings of the set of node indices $\{1, ..., n\}$ as elements. It will be shown in this section that learning on a set actually can be seen as a particular case of learning on graphs.

A directed graph representing a point cloud structure with nodes $\mathcal{V} = \{1, \ldots, n\}$ and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ can be written as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Sets can be seen as a special case of a graph where the domain is only the set of nodes $\mathcal{V}$, and the edges are just the empty set $\mathcal{E} = \varnothing$. The features of a node i that are used for learning on can be written as $\mathbf{x}_i \in \mathbb{R}^k$. They can be stacked together into a $n \times k$ node feature matrix $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)^T$, so that each row of $\mathbf{X}$ corresponds to a feature vector $\mathbf{x}_i$.

It is desirable that the result of a function f acting on the data is independent of the node ordering. More formally, $f(\mathbf{X})$ is *permutation invariant* if, for all $n \times n$ shaped permutation matrices $\mathbf{P}$ it holds that $f(\mathbf{PX}) = f(\mathbf{X})$. A permutation matrix is a square binary matrix that has exactly one entry valued 1 in each row and column, and zeros elsewhere.

For all cases where predictions are needed on node-level instead of set-level, $f(\mathbf{X})$ must be *permutation equivariant* to be able to make predictions. That is the case when $\mathbf{F}(\mathbf{PX}) = \mathbf{PF}(\mathbf{X})$ holds for all permutation matrices $\mathbf{P}$. The bold notation for $\mathbf{F}(\mathbf{X})$ emphasizes that it outputs node-wise vector features, and is thus a matrix-valued function. An equivariant set function can be thought of as transforming each node input $\mathbf{x}_i$ into a latent vector $\mathbf{h}_i = \phi(\mathbf{x}_i)$:

$$\mathbf{F}(\mathbf{X}) = \begin{bmatrix} — & \phi(\mathbf{x}_1) & — \\ — & \phi(\mathbf{x}_2) & — \\ & \vdots & \\ — & \phi(\mathbf{x}_n) & — \end{bmatrix} \tag{5.7}$$

where ϕ is a function applied in isolation to every node. Stacking the node-wise transformations together as in Eq. (5.7) produces a latent space matrix with all transformed features. In practice, ϕ consists of one or many dense layer(s) with shared weights, meaning that when a neural network is trained, the same layers are applied to every node.

Following the geometric blueprint presented in Section 5.3, a general formula for learning on sets can be written as:

$$f(\mathbf{X}) = \rho\left(\bigoplus_{i \in \mathcal{V}} \phi(\mathbf{x}_i)\right). \tag{5.8}$$

The outputs of ϕ are aggregated by $\bigoplus$ denoting some permutation invariant pooling operator (such as sum, average or max) if predictions on set-level are desired. After aggregating the output an additional tail function ρ (e.g. an MLP) is applied that maps to the learnable target.

The blueprint for learning on graphs can be derived in similar fashion. Instead of a set of nodes the full graph $\mathcal{G}$ is considered. The edges can be represented with an $n \times n$

adjacency matrix $\mathbf{A}$, such that:

$$a_{ij} = \begin{cases} 1 & (i,j) \in \mathcal{E} \\ 0 & \text{otherwise.} \end{cases} \tag{5.9}$$

The notion of permutation invariance and equivariance can be generalized to graphs. Both the rows and the columns of $\mathbf{A}$ need to be appropriately permuted when applying permutation matrix $\mathbf{P}$, which amounts to $\mathbf{PAP}^T$. Updated definitions of suitable functions $f(\mathbf{X}, \mathbf{A})$ over graphs can be written as $f(\mathbf{PX}, \mathbf{PAP}^T) = f(\mathbf{X}, \mathbf{A})$ for invariance and $\mathbf{F}(\mathbf{PX}, \mathbf{PAP}^T) = \mathbf{PF}(\mathbf{X}, \mathbf{A})$ for equivariance.

On sets, locality was enforced by applying functions to every node in isolation. For sets it is enough to specify the features and assume the domain to be fixed. Graphs on the other hand have a broader context, and invariance to domain deformation (*geometric stability*) needs to be taken into account. A node's immediate neighborhood can be used to enforce locality in graphs. For a node i, its closest neighbors are:

$$\mathcal{N}_i = \{j : (i,j) \in \mathcal{E} \vee (j,i) \in \mathcal{E}\}. \tag{5.10}$$

Using the notion of neighborhood the set of features in the neighborhood can be extracted as:

$$\mathbf{X}_{\mathcal{N}_i} = \{\mathbf{x}_j : j \in \mathcal{N}_i\}. \tag{5.11}$$

To construct permutation equivariant functions on graphs, a local function $\phi(\mathbf{x}_i, \mathbf{X}_{\mathcal{N}_i})$ that operates over the set of features must be defined. The function ϕ takes the features of node i as well as the entire set of features in the neighborhood and updates the features of node i based on this locality. The graph update operation is illustrated in Fig. 5.6.

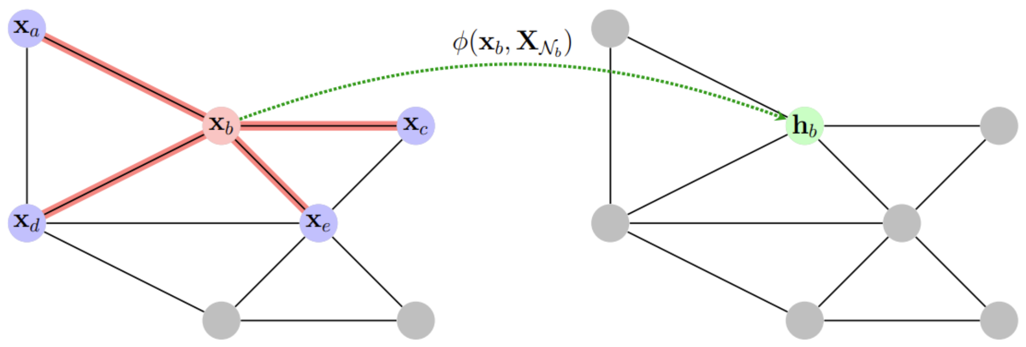


Figure 5.6: A permutation-equivariant function operating on a graph by applying a permutation-invariant function ϕ to every neighborhood. In the picture ϕ is applied to the features $\mathbf{x}_b$ of node b and the set of its neighboring features $\mathbf{X}_{\mathcal{N}_i} = \{\mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_c, \mathbf{x}_d, \mathbf{x}_e\}$. Applying the function ϕ to every node's neighborhood results in a matrix of latent features $\mathbf{H} = \mathbf{F}(\mathbf{X}, \mathbf{A})$ [31].

With this information at hand a permutation equivariant function $\mathbf{F}$ can be constructed as follows:

$$\mathbf{F}(\mathbf{X}, \mathbf{A}) = \begin{bmatrix} — & \phi(\mathbf{x}_1, \mathbf{X}_{\mathcal{N}_1}) & — \\ — & \phi(\mathbf{x}_2, \mathbf{X}_{\mathcal{N}_2}) & — \\ & \vdots & \\ — & \phi(\mathbf{x}_n, \mathbf{X}_{\mathcal{N}_n}) & — \end{bmatrix}. \tag{5.12}$$

Importantly, the output of ϕ must be independent of the ordering of the nodes in the neighborhood $\mathcal{N}_i$ for $\mathbf{F}$ to be permutation equivariant. The overall computation of $\mathbf{F}$ is in practice referred to as a layer in a GNN.

When learning on graphs the latent features $\mathbf{h}_i$ of a node i can be constructed by applying some learnable function $\psi(\mathbf{x}_i, \mathbf{x}_j)$ on the neighboring features and aggregating the result over the neighborhood $\mathcal{N}_i$ using a permutation invariant operation $\bigoplus$. Another learnable function ϕ is then further applied that updates the features of node i to a latent feature vector $\mathbf{h}_i$:

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} \psi(\mathbf{x}_i, \mathbf{x}_j)\right). \tag{5.13}$$

GNNs can be classified into different flavors depending on how edge interactions $\psi(\mathbf{x}_i, \mathbf{x}_j)$ are computed as depicted in Fig. 5.7. A simple and well studied flavor is *convolutional* GNNs with $\psi(\mathbf{x}_i, \mathbf{x}_j) = c_{ij}\psi(\mathbf{x}_j)$. Here c_{ij} is a constant specifying the importance of node i to node j's representation, often directly dependant on the entries in the adjacency matrix $\mathbf{A}$ that represents the graph structure. Another type of GNN is the *attentional* flavor with $\psi(\mathbf{x}_i, \mathbf{x}_j) = a(\mathbf{x}_i, \mathbf{x}_j)\psi(\mathbf{x}_j)$ where a is a learnable importance coefficient. The attentional flavor allows for modeling more complex interactions within neighborhoods by computing scalar-valued quantities across the edges. However, most information can be extracted by the *message-passing* GNN flavour where vector-valued

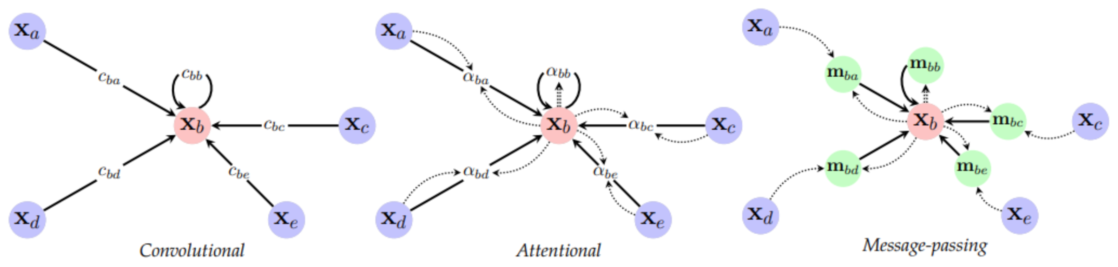


Figure 5.7: Current graph neural networks can be divided into three flavors: 1. convolutional for which sender node features are multiplied with a constant c_{ij}, 2. attentional where the factor is implicitly computed using an attention mechanism of the receiver over the sender $a_{ij} = a(\mathbf{x}_i, \mathbf{x}_j)$ and 3. message-passing where messages in vector form are calculated based on both the sender *and* the receiver $\mathbf{m}_{ij} = \psi(\mathbf{x}_i, \mathbf{x}_j)$ [31].

messages are computed across edges, which translates to using $\psi(\mathbf{x}_i, \mathbf{x}_j)$ as is. Because the choice of ψ increase in generality one can note that the flavors can be represented by each other, such that *convolutional* $\subseteq$ *attentional* $\subseteq$ *message-passing*.

Latent features can be used in an equivariant setting directly to make predictions on node level as seen in Fig. 5.8. In that case a learnable function ρ is applied immediately on the node features $\mathbf{h}_i$. A second option is to do predictions on links in the graph. Then, information about a node i, its neighbor j and edge features $\mathbf{e}_{ij}$ that might exist between them, is used by ρ. Finally, graph-level predictions can be made by performing a permutation invariant aggregation over all node features and applying a permutation invariant tail function. This can be written as:

$$f(\mathbf{X}, \mathbf{A}) = \rho\left(\bigoplus_{i\in\mathcal{V}} \phi(\mathbf{x}_i, \mathbf{X}_{\mathcal{N}_i})\right) \tag{5.14}$$

where $\bigoplus_{i\in\mathcal{V}}$ is the global pooling operation. The additional input $\mathbf{x}_i$ to ϕ represents an optional skip-connection, which is often very useful. Eq. (5.14) is a manifestation of the geometric deep learning blueprint since all the necessary elements are present: node-wise $\mathfrak{G}$-equivariant layers ϕ with local pooling, and a $\mathfrak{G}$-invariant readout layer at the end.

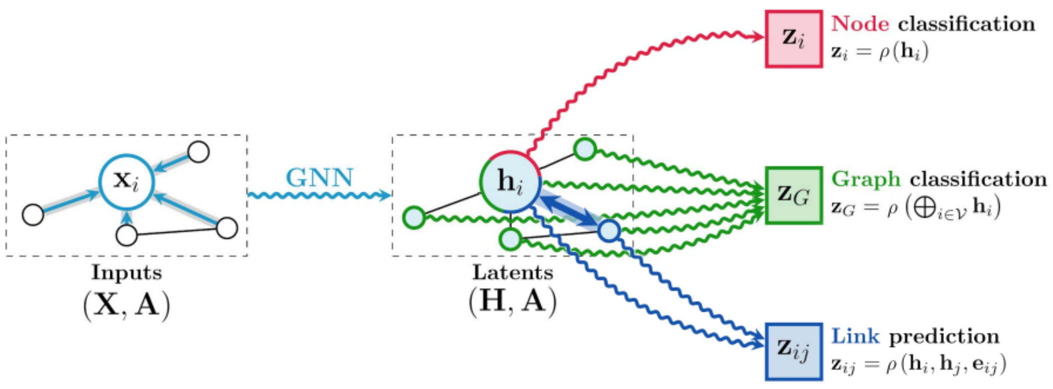

$\mathbf{x}_i$
GNN
$\mathbf{h}_i$
Inputs
$(\mathbf{X}, \mathbf{A})$
Latents
$(\mathbf{H}, \mathbf{A})$
$\mathbf{z}_i$
Node classification
$\mathbf{z}_i = \rho(\mathbf{h}_i)$
$\mathbf{z}_G$
Graph classification
$\mathbf{z}_G = \rho\left(\bigoplus_{i\in\mathcal{V}} \mathbf{h}_i\right)$
$\mathbf{z}_{ij}$
Link prediction
$\mathbf{z}_{ij} = \rho(\mathbf{h}_i, \mathbf{h}_j, \mathbf{e}_{ij})$

Figure 5.8: Having run a GNN on a graph-based dataset provides you with a learned latent feature matrix $\mathbf{H}$ that can utilized in different ways depending on the application. It is possible to do predictions on node level directly on the latent features $\mathbf{h}_i$, or you can aggregate the latent features into global graph level outputs, $\bigoplus_{i\in\mathcal{V}} \mathbf{h}_i$, and do predictions on them. A final option is to use node and link information, $\mathbf{h}_i, \mathbf{h}_j, \mathbf{e}_{ij}$, for link level predictions [34].

6. Jet energy regression

Machine learning is widely used within the high energy physics community for analyzing the data that comes out of the collisions. However, because the resulting particles branch off in different ways the resulting data is in the form of variable-length lists with no intrinsic ordering. Traditional machine learning models are not well suited for dealing with such data, but there has been progress on that front with learning on *point clouds* in computer vision. In the case of particle physics the underlying data comprised of unordered sets can be seen as *particle clouds* in the (η, ϕ)-plane analogous to the point clouds used to represent a 3D image in the (x, y, z)-plane for example. Neural networks for classifying, or tagging, jets have been trained successfully on particle clouds. The problem in this thesis is not so far off from jet tagging. The aim is to predict particle-level jets' transverse momentum, which is a continuous variable, using reconstructed data. This means that instead of performing classification, it is a regression task. However, the same type of data and model architectures used for jet tagging are applicable to this problem.

6.1 Dataset

The simulated event samples used in this study were produced using the Pythia 8 event generator with the CUETP8M1 parameter tune. The QCD dijet events were simulated at a center of mass energy of 13 TeV and binned according to H_T, which is the sum of generator level hadron transverse energies. The detector simulation is based on the state of CMS during the summer 2016 when Run II took place. Jets were reconstructed from PF candidates and clustered using the anti-k_T algorithm [26] with $R = 0.4$. The CHS method was used for pileup mitigation. The program that produced the jet samples[2] is a fork of previous ML JEC studies by A. Popov [35], and it has been run within the CMS Software framework [36]. The dataset has 14 million jets in total sharded into Mini-AOD [37] compressed ROOT [38] files with 100 thousand jets

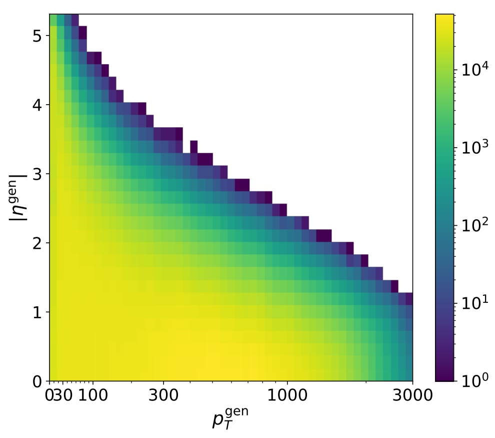
[2] `gitlab.cern.ch/dholmber/ml-jec-vars`

each. The data is divided into a training set with 60% of the jets, and validation and test sets with 20% of the jets each.

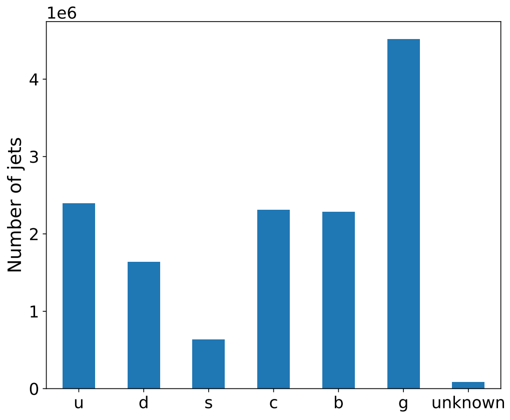
The data is downsampled in overpopulated regions (e.g. way too many low p_T jets) of the parameter ($p_\mathrm{T}^\mathrm{gen}$, η^gen, flavor) space. The dataset has fixed proportions of different jet flavors such that the proportions of b, c, uds and g jets are $1:1:2:2$ respectively. In addition, the distribution of data has the same shape for all jet flavors. It is made flat in (p_T, η) at low p_T and steeply falling at high p_T by sampling from a probability density function based on the hyperbolic cosine function:

$$h_T(p_\mathrm{T}, \eta) \propto \left[\cosh\frac{p}{p_\mathrm{ref}} \cdot \cosh\eta\right]^{-1} \tag{6.1}$$

where $p = p_\mathrm{T}\cosh\eta$ is the magnitude of jet momentum. The parameter p_ref is chosen to be 500 GeV based on trials and the available statistics at high p [39]. The resulting distribution of data is displayed in Fig. 6.1.



(a) Spectrum of jets in (p_T, η) bins



(b) Number of jets per flavor

Figure 6.1: Distribution of jets used in this study. The heatmap to the left shows the number of jets as a function of p_T and η, and to the right the number of jets of different flavors are displayed in bar diagram.

The dataset includes jet-level data, PF candidates, and secondary vertices (SV). Jet-level data consists of global features describing a jet. In practice, there are multiple particles making up a jet. The PF candidate data describes the particles inside a jet, and the SV data depicts the interaction points of daughter particles in a jet. The PF candidates can further be divided into charged and neutral particles. The division between charged and neutral constituents can be beneficial since they behave differently in the detector. The sets of particles in the PF candidate collections are up to 64 elements in size, whereas the SV collection consist of smaller sets with up to 16 elements each. A jet with many constituents is shown using detector coordinates in Fig. 6.2. The plot visualizes what is known as a particle cloud in the (η, ϕ)-plane.

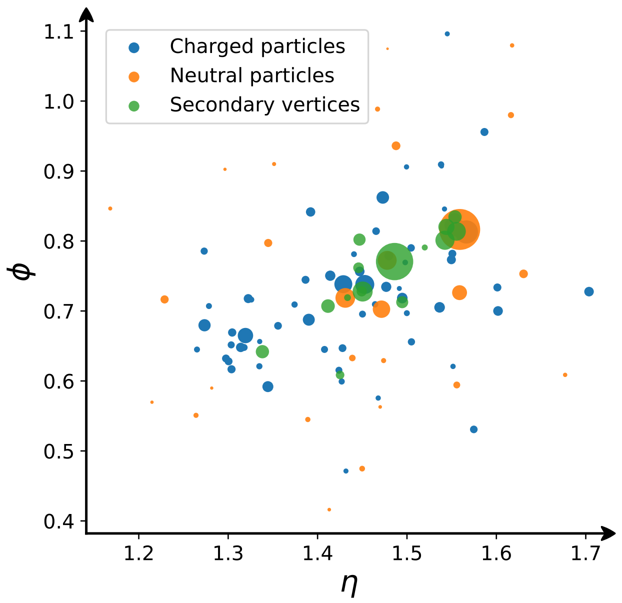


Figure 6.2: Constituents of a large jet with 64 charged particles, 27 neutral particles and 15 secondary vertices shown as a particle cloud. The size of the markers is proportional to the constituent's p_{T}.

Global observables in the dataset are listed in Table 6.1, and jet constituent features are listed in Table 6.2. The feature tables include all variables, even synthetically created ones. All of them are used for training neural networks as presented in a later section.

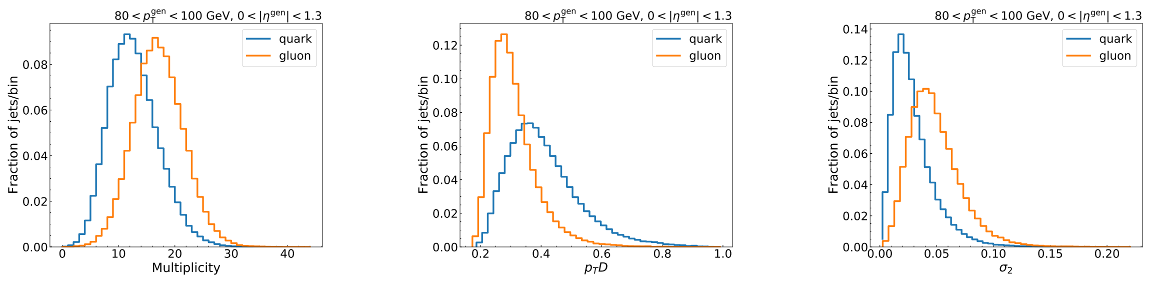
Variable	Description
p_{T}	Transverse momentum of the jet
η	Pseudorapidity of the jet
ϕ	Azimuthal angle of the jet
ρ	Median angular p_{T} density
m	Jet mass
A	Jet catchment area
n_{PV}	Number of good primary vertices
$p_{\mathrm{T}}D$	Jet fragmentation distribution
σ_2	Jet minor angular opening
multiplicity	Jet constituent multiplicity

Table 6.1: Jet-level input features.

Variable	Description	Charged PF	Neutral PF	SV
p_{T_i}	Transverse momentum of a jet constituent	x	x	x
η_i	Pseudorapidity of a jet constituent	x	x	x
ϕ_i	Azimuthal angle of a jet constituent	x	x	x
Δp_{T_i}	Fractional p_T of a jet constituent	x	x	x
$\Delta\eta_i$	Fractional pseudorapidity of a jet constituent	x	x	x
$\Delta\phi_i$	Fractional azimuthal angle of a jet constituent	x	x	x
d_{xy}	Distance in the xy-plane from the primary vertex	x		
d_{xy}^sig	d_{xy} significance	x		
d_z	Distance in z from the primary vertex	x		
χ^2_trk	Normalized track χ^2	x		
n_hits	Number of hits	x		
$n_\mathrm{pixel\ hits}$	Number of pixel hits	x		
lost hits	Lost inner hits	x		
hcal frac	HCAL energy fraction		x	
d_{SV}	Flight distance			x
d_{SV}^sig	Flight distance significance			x
n_tracks	Number of tracks			x
id	Particle id code	x	x	x
PV quality	Primary vertex association quality	x		

Table 6.2: Jet constituent input features.

In addition to the jet features that are readily available in the dataset as is, such as jet p_T, η, ϕ, mass, and so on, some feature engineering is applied producing new synthetic features. To help with light quark/gluon discrimination three features (Fig. 6.3) are created: PF candidate multiplicity, jet transverse momentum distribution variable $p_\mathrm{T}D$, and jet minor axis σ_2.



(a) Jet constituent multiplicity **(b)** Jet p_T distribution variable **(c)** Jet ellipse minor axis

Figure 6.3: Features used for discriminating quarks and gluons

The multiplicity is computed as the number of PF candidates with $p_\mathrm{T} > 1$ GeV in a jet. Because gluons have a higher color charge than light quarks, they produce more and softer particles [40]. Gluons therefore have higher multiplicity than quarks which makes it a good feature for discriminating between the two types of jets.

The transverse momentum distribution variable:

$$p_{\mathrm{T}}D = \frac{\sqrt{\sum_i p_{T,i}^2}}{\sum_i p_{T,i}} \tag{6.2}$$

expresses the difference in p_{T} distribution between quarks and gluon. For jets made of one particle carrying all momentum $p_{\mathrm{T}}D \to 1$, whereas $p_{\mathrm{T}}D \to 0$ for jets with an infinite amount of particles carrying the total momentum. Since gluons produce softer jets in general, $p_{\mathrm{T}}D$ will be lower for them on average.

It is well motivated to approximate a jet as an ellipse because of the conical shape of clustered jets. A certain feature of the ellipse called the minor axis, σ_2, can be used to distinguish how collimated, or wide, a jet is in the (η, ϕ) plane. Gluon hadronization is expected to produce wider jets than quark hadronization, so σ_2 will in general be larger for gluon jets.

Other useful features created from already existing ones are relative p_{T}, η and ϕ. They represent the proportion of a certain jet-level observable contained by a specific particle in the PF candidates which can be useful information for an ML model. The relative features are defined as follows:

$$\Delta p_{\mathrm{T}} = p_{\mathrm{T}}^{\mathrm{pf}} / p_{\mathrm{T}}^{\mathrm{jet}} \tag{6.3}$$

$$\Delta\eta = \mathrm{sgn}(\eta^{\mathrm{jet}})(\eta^{\mathrm{pf}} - \eta^{\mathrm{jet}}) \tag{6.4}$$

$$\Delta\phi = (\phi^{\mathrm{pf}} - \phi^{\mathrm{jet}} + \pi) \bmod 2\pi - \pi. \tag{6.5}$$

There are some categorical features in the dataset, where each category is represented as a unique integer. To emphasize the difference between categories, categorical features are *one-hot encoded*, meaning that orthogonal vectors are created for every category. For example, neutral particles have the following IDs: 1 = down quark (d), 2 = up quark (u), 22 = photon (γ) and 130 = long-lived neutral kaon (K_L^0) [41]. When encoded, these particle IDs would map to the vectors shown in the table below.

d	1	0	0	0
u	0	1	0	0
γ	0	0	1	0
K_L^0	0	0	0	1

Table 6.3: One-hot encoded neutral particle IDs.

6.2 Target and loss function

The reconstructed jets' transverse momentum are corrected towards particle-level using basic kinematic quantities as laid out in Sec. 4.3. ML regression can improve those corrections further. The target of the regression is to predict $p_\mathrm{T}^\mathrm{gen}$ knowing $p_\mathrm{T}^\mathrm{reco}$ as well as other reconstructed features. However, to avoid having a wide and skewed target distribution, the correction factor $p_\mathrm{T}^\mathrm{gen}/p_\mathrm{T}^\mathrm{reco}$ is used as target instead of $p_\mathrm{T}^\mathrm{gen}$. Furthermore, the logarithm is taken of the correction factor yielding the final training target y as:

$$y = \log\left(\frac{p_\mathrm{T}^\mathrm{gen}}{p_\mathrm{T}^\mathrm{reco}}\right) \tag{6.6}$$

which gives a nice centralized distribution as seen in Fig. 6.4. To then get the corrected transverse momentum $p_\mathrm{T}^\mathrm{corr}$ the predicted correction factor $\exp(y)$ must be multiplied by $p_\mathrm{T}^\mathrm{reco}$. From there one can define the per-jet response as:

$$R = \frac{p_\mathrm{T}^\mathrm{corr}}{p_\mathrm{T}^\mathrm{gen}}. \tag{6.7}$$

R should ideally hover around unity, and more importantly, have a good resolution. Mean absolute error (MAE) is used as loss function. To avoid spikes in loss during

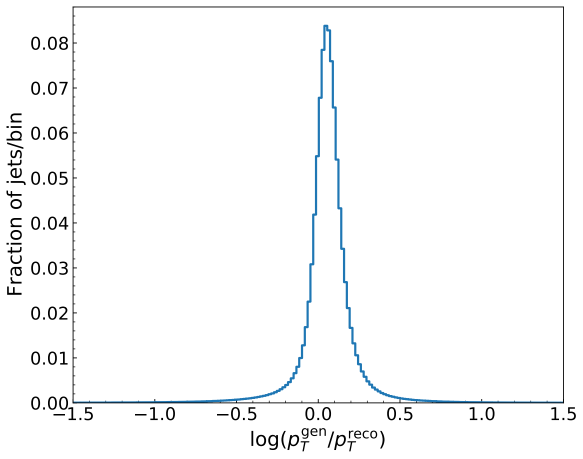


Figure 6.4: The logarithm of the transverse momentum correction factor will be used as regression target in this study.

training, outliers with a target value smaller than -1 or larger than 1 are ignored by the ML models. The loss function becomes:

$$L = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}_i| I_{|y_i|<1} \tag{6.8}$$

where N is the total number of samples, y is the true target value, and $\hat{y}$ is the predicted target value.

MAE showed better results than the more commonly used mean squared error (MSE) loss during trial runs, which is why it was chosen. Here follows a brief explanation on how these two losses compare, and a motivation for choosing MAE in this case. A loss function can be seen as a *functional* rather than just a function. When a sufficiently powerful network learns, it can be thought of as choosing a function $\hat{y} = \hat{f}(x)$ from a wide class of functions. Different cost functions can give different statistics depending on what function minimizes it. For example, the minimum of mean squared error (MSE) lies on the function that maps x to the expected value of y given x. In the case of MAE the function minimizing $\mathbb{E}|y - \hat{y}|$ is $\hat{f}(x) = \text{Median}(y \mid X = x)$ [32]. Predicting the median can be seen as beneficial since it is a robust measure of central tendency, meaning that regardless of whether the conditional distribution is asymmetric the fit is unbiased. MAE gives a smaller importance to outliers compared to the MSE loss. If the costs associated with errors increase linearly for the application, in this case errors for $p_{\mathrm{T}}^{\mathrm{gen}}$ predictions, then MAE is likely the sounder choice. A previous study of b-jet energy regression [4] also concluded that a cost function with reduced sensitivity to the tails of the target distribution is preferred.

6.3 Models

The models that will be used for performing jet energy regression must be able to deal with the data structure of the PF candidates, which is variable length unordered sets. A graph can further be constructed from the constituents by using η and ϕ as coordinates, thus forming a particle cloud. Two novel architectures that can be trained on such data are: 1. the set based *Deep Sets* architecture [6], and 2. the graph based *Dynamic Graph Convolutional Neural Network* (DGCNN) [42] architecture. These network types have been adopted by experimental particle physicists to improve the accuracy of jet tagging, which is a classification task in machine learning terms. They have proposed their own versions of the architectures, *Particle Flow Network* (PFN) [43] and *ParticleNet* [7] respectively.

Improving the jet energy resolution is a regression task rather than a classification one, however, the same type of data can be used for training in both of these tasks. In addition, both PFN and ParticleNet performed very well in classification, ParticleNet even reaching state of the art. Both of those reasons make PFN and ParticleNet compelling choices for jet energy regression. The comparison between the two architectures is also an interesting one since they lie at the opposite end of the complexity spectrum of architectures that can be trained on particle clouds. PFN is a purely set based model, whereas ParticleNet calculates a graph using a costly k nearest neighbor (k-NN) operation, and furthermore belongs to the class of message-passing GNNs with inherently computationally expensive edge features.

6.3.1 Mutual architectural choices

Both PFN and ParticleNet use the same activation function, the *rectified linear unit* (ReLU):

$$\mathrm{ReLU(x)} = \max(0, x) \tag{6.9}$$

where x is the input to a neuron [44]. ReLu is a computationally very simple and fast activation function. Additionally, network units are sparsely activated in the beginning when weights are commonly initialized uniformly around zero. Sparsity can help to disentangle factors explaining variations in the data [45]. The output layer in an unbounded regression problem can be without an activation function since nonlinearities are only required in hidden layers in the network. The output is then said to have *linear activation*, which is the case for the models in this study. This differs from how both models are originally implemented. They are designed to deal with classification problems and each have a softmax activation function as readout layer. The networks are named PFN-r and ParticleNet-r to signify the difference from the original models.

The *He-normal* initializer is used for weight initialization in both models since it is derived for, and proven to work well with rectifier activations [46]. It draws samples from a truncated normal distribution centered around zero with standard deviation equal to $\sqrt{2/n_l}$, where n_l is the number of input units of a layer.

Batch normalization (BatchNorm) [47] can improve the training performance of deep neural networks and is used for both models in this study. It works by normalizing the values in a mini-batch $\mathcal{B} = \{x_{1 \dots m}\}$ using the mini-batch mean $\mu_{\mathcal{B}}$ and standard deviation $\sigma^2_{\mathcal{B}}$:

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \tag{6.10}$$

$$y_i = \gamma \hat{x}_i + \beta. \tag{6.11}$$

A small ϵ is added to the denominator for numerical stability. The normalized activation is represented by $\hat{x}_i$, which has zero mean and unity variance if ϵ is omitted. The internal output $\hat{x}_i$ is multiplied by a learned scale parameter γ and then summed with a learned shift parameter β. Hence, it is unnecessary to add a bias term to linear layers in the network since BatchNorm already has one built in. BatchNorm is normally applied directly after linear layers since those are more likely to have symmetric non-sparse distributions, compared to after activations have been applied.

For neural networks with many layers the gradients computed by backpropagation using the chain rule can become increasingly smaller leading to slow weight updates in early layers. BatchNorm can help with this via its normalization step that fixes the means and variances of layer inputs. This allows the gradient to flow through more easily since the gradient dependence on parameter scales is reduced. When using BatchNorm a higher learning rate can be used, speeding up training without the risk of vanishing or diverging gradients. Another phenomenon that affects deep neural networks is *internal covariate shift*. As the parameters of a layer change, the distribution of inputs to the next layer change accordingly, and so on. These shifts in input distributions are amplified as they propagate within the network. BatchNorm is proposed to reduce such unwanted shifts producing more reliable models and speeding up training further. Lastly, because random values are included in a minibatch at each step, hidden units in BatchNorm are multiplied by values that fluctuate a little. This noise leads to BatchNorm having a slight regularization effect, making the models less prone to overfitting.

Both models use the same iterative optimization algorithm for finding the minimum of the loss function. The optimizer, named *Adaptive Moment Estimation* (Adam) [48], is based on a stochastic approximation of gradient descent. This is because the gradient is calculated on one subset, or *batch*, of the full dataset at a time. The gradient is usually calculated in batches in deep learning applications because the training data is usually too large to fit into a computer's working memory all at once.

To improve upon vanilla stochastic gradient descent, the learning rate is adapted for each parameter in the Adam algorithm. The first moment (mean) as well as the second moment (uncentered variance) of the gradients are used to calculate the learning rate for a weight. Empirical results have shown that Adam compares favorably to other optimization methods, making it a very common optimizer choice.

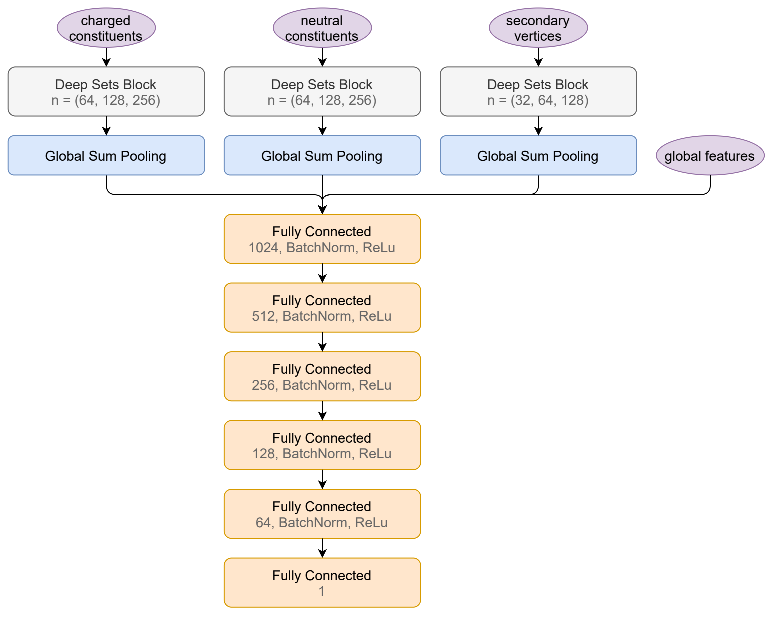
6.3.2 Particle Flow Network

PFN is a Deep Sets network used for jet flavor tagging. As seen in Sec. 5.4, the model for learning on sets is a simplified version of graph learning with an empty set of edges. Deep Sets and consequently PFN adopts the blueprint for learning on sets directly as is, with latent features as $\mathbf{h}_i = \phi(\mathbf{x}_i)$, where $\mathbf{x}_i \in \mathbb{R}^F$ is a row with F separate feature values. The output of ψ consists of a set with n latent particles. The complete network, with a permutation invariant tail for making global predictions, can be described by:

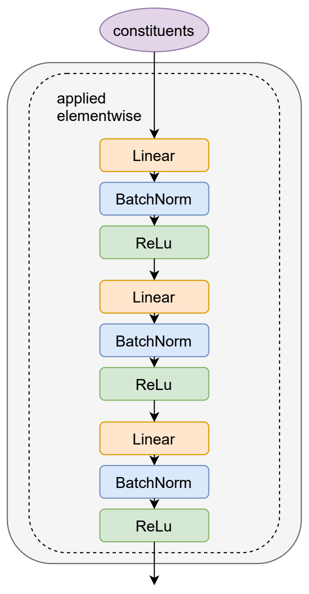
$$f(\mathbf{X}) = \rho\left(\sum_{i\in\mathcal{V}} \phi(\mathbf{x}_i)\right). \tag{6.12}$$

Summation was chosen as the global pooling operation in line with the PFN paper. It is a perfectly valid choice since the output is independent of the ordering of the inputs. The equation above describes the PFN network applied to only one set of particles in some constituent collection, however PFN-r takes multiple collections as inputs separately.

The full PFN-r architecture is illustrated in Fig. 6.5a, and the layers operating on individual particles in set based collections is shown in Fig. 6.5b. The network passes charged and neutral PF candidates as well as secondary vertices through blocks based on Eq. (6.12). The inner mapping ϕ consists of three dense layers with BatchNorm and ReLu as activation function. The Deep Sets blocks for PF candidates are trained with more units (PFN-r: 64, 128, 256, PFN-r Lite: 16, 32, 64) compared to the lower multiplicity SV collection (PFN-r: 32, 64, 128, PFN-r Lite: 8, 16, 32). Global sum pooling is applied on the Deep Sets output. The tensors can then be concatenated with jet-level features since they have the same number of rows at that point. Another MLP block ρ is applied to them, again with linear, BatchNorm and ReLU layers. The number of units are (1024, 512, 256, 128, 64) for PFN-r and (128, 64, 32, 16, 8) for PFN-r Lite. Lastly, a linear output layer with 1 unit applied mapping to the regression target.


charged constituents
neutral constituents
secondary vertices
Deep Sets Block n = (64, 128, 256)
Deep Sets Block n = (64, 128, 256)
Deep Sets Block n = (32, 64, 128)
Global Sum Pooling
Global Sum Pooling
Global Sum Pooling
global features
Fully Connected 1024, BatchNorm, ReLu
Fully Connected 512, BatchNorm, ReLu
Fully Connected 256, BatchNorm, ReLu
Fully Connected 128, BatchNorm, ReLu
Fully Connected 64, BatchNorm, ReLu
Fully Connected 1

(a) PFN-r



(b) Deep Sets block

Figure 6.5: The structure of the Particle Flow Network, as implemented in this thesis.

6.3.3 ParticleNet

ParticleNet is a GNN that uses relative η and ϕ as initial coordinates for vertices in a graph. An *Euclidean distance matrix* is created from pairwise distances between points in the ($\Delta\eta$, $\Delta\phi$) - plane. The element of the distance matrix are given by the squares of the distances between the points $d_{ij} = ||\mathbf{p}_i - \mathbf{p}_j||^2$. In total there are n points per set, and they belong to $\mathbb{R}^2$ since there are two coordinates. The k-NN algorithm is used to construct edges in the graph as connections between each point and its k closest points. Since there exists a local patch for every point, convolution operations can be run on them.

Messages between every point and its neighbors are calculated using an *asymmetric edge function* $\psi(\mathbf{x}_i, \mathbf{x}_j - \mathbf{x}_i)$, so that information about both global shape structure and the local k-hop neighborhood is included. The edge function ψ is implemented as an MLP consisting of three linear layers each with BatchNorm and ReLU activation applied to the edge features. In practice, the MLP is implemented via graph convolution with number of channels corresponding to the number of units in each linear transformation layer, or in other words, the output dimensionality of the feature vector at every node. Because of the convolutions, the learnable parameters in ϕ are shared for all nodes in the graph. A channel-wise symmetric operation chosen to be the mean, aggregates the edge features for every vertex. This concludes the *EdgeConv* operation [42] given as the second input to ϕ:

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \frac{1}{k}\sum_{i\in\mathcal{N}_i^k} \psi(\mathbf{x}_i, \mathbf{x}_j - \mathbf{x}_i)\right). \tag{6.13}$$

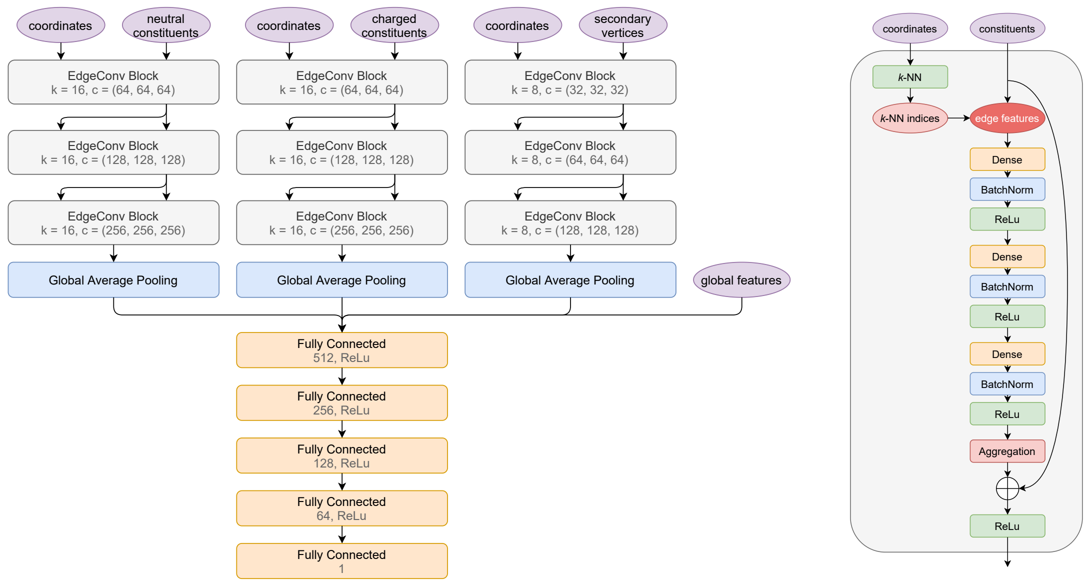
A skip connection runs in parallel with the EdgeConv operation. The skip connection is a learnable linear mapping that works as a shortcut, and in theory the network can be seen as learning deviations from that identity layer. It is experimentally validated that a strong linear component is beneficial for the convergence of deep networks [49]. The shortcut is given as the right input in Eq. (6.13). The function ϕ is implemented as the ReLU activation function.

The graph is dynamically updated, hence the name *Dynamic Graph* CNN. A new pairwise distance matrix is computed on the feature space resulting from an EdgeConv block. After the first EdgeConv block the distance elements d_{ij} are calculated from points with dimensionality equal to the number of features: $\mathbf{p}_i, \mathbf{p}_j \in \mathbb{R}^F$. All vertices have their neighborhoods recalculated with k-NN based on the new distance matrix, creating an entirely new graph for the next EdgeConv operation. Dynamic graph updates improves performance since proximity in feature space differs from proximity in model input, and updating the graph resolves that [42].

To perform jet-level regression global average pooling is used to aggregate the learned features over all n particles in the cloud. Then, like for PFN-r, the output is concatenated with jet-level features and given as input to an MLP with linear, BatchNorm and ReLU layers. The very last layer is a fully connected linear mapping to the target. The MLP and the output layer are collectively expressed as ρ:

$$f(\mathbf{X}, \mathbf{A}) = \rho\left(\frac{1}{n}\sum_{i\in\mathcal{V}}\phi(\mathbf{x}_i, \mathbf{X}_{\mathcal{N}_i^k})\right). \tag{6.14}$$

The adjacency matrix $\mathbf{A}$ given as input to the full network f are the indices of features in a vertex i:s neighborhood produced by k-NN, which are updated for every EdgeConv block. The neighborhood features themselves are expressed as $\mathbf{X}_{\mathcal{N}_i^k}$. One thing to note about equation (6.14) is that it shows global pooling and a final learnable function applied on one set of features with n rows from one constituent collection. In the actual network, ρ takes more collections as inputs. Jet-level data, SV, charged and neutral PF candidates are passed as separate inputs as shown in Fig. 6.6.


coordinates
neutral constituents
coordinates
charged constituents
coordinates
secondary vertices
EdgeConv Block k = 16, c = (64, 64, 64)
EdgeConv Block k = 16, c = (64, 64, 64)
EdgeConv Block k = 8, c = (32, 32, 32)
EdgeConv Block k = 16, c = (128, 128, 128)
EdgeConv Block k = 16, c = (128, 128, 128)
EdgeConv Block k = 8, c = (64, 64, 64)
EdgeConv Block k = 16, c = (256, 256, 256)
EdgeConv Block k = 16, c = (256, 256, 256)
EdgeConv Block k = 8, c = (128, 128, 128)
Global Average Pooling
Global Average Pooling
Global Average Pooling
global features
Fully Connected 512, ReLu
Fully Connected 256, ReLu
Fully Connected 128, ReLu
Fully Connected 64, ReLu
Fully Connected 1
coordinates
constituents
k-NN
k-NN indices
edge features
Dense
BatchNorm
ReLu
Dense
BatchNorm
ReLu
Dense
BatchNorm
ReLu
Aggregation
ReLu

(a) ParticleNet-r **(b)** EdgeConv block

Figure 6.6: The ParticleNet-r architecture. The complete network can be seen on the left side, and the structure of the EdgeConv block is illustrated on the right.

Two ParticleNet-r versions are trained for comparison, one with many parameters as well as a light version with less parameters. The number for k is set to 16 for both charged and neutral PF candidates, and 8 for the smaller SV collection in both ParticleNet-r and ParticleNet-r Lite. The number of channels in each of ParticleNet-r's EdgeConv blocks for PF candidate are (64, 64, 64), (128, 128, 128) and (256, 256, 256). For SV they are set as (32, 32, 32), (64, 64, 64) and (128, 128, 128). In ParticleNet-r Lite there are also three EdgeConv blocks, but with two layers each. EdgeConv blocks for PF candidates have (16, 16), (32, 32) and (48, 48) units each, whereas the corresponding SV units are (8, 8), (16, 16) and (24, 24). Lastly, the MLP that makes up the tail of the network has (512, 256, 128, 64) units in ParticleNet-r and (64, 32, 16, 8) units in ParticleNet-r Lite.

6.4 Implementation and training

The whole training pipeline was implemented using the open-source TensorFlow framework [50]. TensorFlow can compute complex tensor operations, crucial to ML. It runs on distributed systems and on multiple different computational devices such as GPU cards. A plethora of built-in ML related algorithms can be accessed and executed using application programming interfaces (APIs) in different programming languages. This work[1] used the Python API to ingest particle data and train jet energy regression models in an efficient manner.

The upper picture in Fig. 6.7 shows a standard ML input pipeline where data is read, preprocessed, and trained on sequentially. However, training can be sped up using TensorFlow's data API [51] which makes it possible to process data in parallel. The bottom picture in the aforementioned figure shows exactly that. There the same steps are executed as in the naive pipeline, but the preprocessing overlap because many CPU cores are working in parallel. In addition to parallel maps, *prefetching* is also used to decrease training time by reading elements from the input dataset to an internal buffer while some other data is still being trained on. The number of cores used by parallel mapping, and the number of elements to prefetch were both automatically tuned in a dynamic fashion at runtime by TensorFlow's data API in the training pipeline used for this study.

Plenty of computing power was available for training the neural networks. The system used for training has an AMD Ryzen Threadripper 3960X CPU chip with 24 cores supporting 2 threads each and an unboosted clock speed of 2.2 GHz. All 48 threads were used in parallel by Tensorflow. Since the input was processed heavily in parallel one might think that the working memory becomes a bottleneck, however 256GB RAM DDR4 was plenty. Even if the RAM is insufficient the TensorFlow pipeline that was used can auto-adjust to available memory. Finally, the most crucial components for effective deep learning are the hardware accelerators, in this case GPUs. The models were trained on two of the latest generation Nvidia GeForce RTX 3090 cards.

The training progresses in *epochs*. An epoch is defined as one pass of the full training

[1] `gitlab.cern.ch/dholmber/jec-gnn`

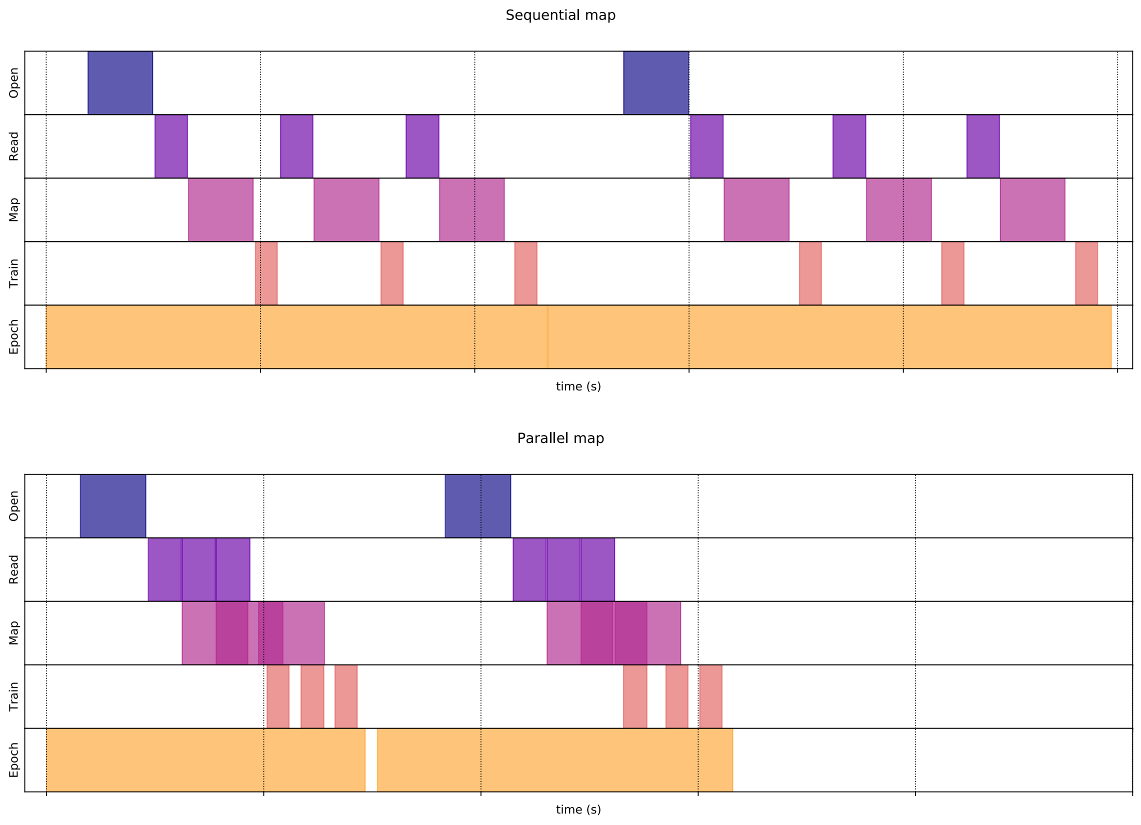


Figure 6.7: Comparison between a naive TensorFlow data pipeline (above), and one running in parallel (below) [52]. The time it takes to finish training a deep learning model is significantly sped up by reading, mapping and training multiple data samples at once.

set through a model. Every epoch consists of many training steps each processing a batch with some number of elements (1024 in this case). After each epoch has finished training the model predicts the target on the validation set. Since the validation set is independent of the training set, the validation loss is a good measure of how a model is performing. Two algorithms, or callbacks, that monitor validation loss to make informed decision regarding training are used for the jet energy regression. Firstly, more accurate minima for the loss function can be found by reducing the learning rate when the validation loss plateaus. In this study, the learning rate was reduced by a factor of five when that happens. The threshold for what counts as an improvement was set as 10^{-4}, and the reduction triggers after five epochs with no improvement. Secondly, training is stopped early to prevent overfitting. The minimum value that the early stopping callback qualifies as an improvement is also set to 10^{-4}. Training is stopped if no improvement is seen after seven epochs, and the model weights for the epoch with the lowest validation loss are restored. It makes sense to have the early stopping callback be slightly more patient than the learning rate reduction callback, because the model can potentially be improved by changing the learning rate.

7. Results

The performance of the deep learning models are evaluated from a few different perspectives. Firstly, it is looked at from the perspective of model complexity. Comparisons of the different architectures presented in the last chapter in terms of computational complexity can be helpful when speed needs to be taken into account in addition to the model's predictive performance. The second metric presented here is the reduction in flavor dependence, meaning that the p_T response should ideally be similar for all flavors. Lastly, and arguably the most important metric, is the improvement in jet energy resolution.

7.1 Model complexity

Table 7.1 shows the number of parameters in each of the neural networks. PFN-r has the most parameters, followed by ParticleNet-r. The light versions of the networks have far fewer parameters, and ParticleNet-r Lite is the one with slightly less of the two. Standard corrections rely on only about a thousand parameters, none of which are trained using machine learning.

Both versions of ParticleNet-r are significantly slower to train due to having more demanding GPU computations. Calculating the graph of neighbors is especially time consuming. The higher complexity is also reflected in both ParticleNet-r models' inference times. PFN-r is over ten times faster for making predictions. The inference time for the coffea [53] implementation of the standard approach is also included for comparison. It has an inference time two orders of magnitude faster than the PFN models.

The loss value for the test set, calculated using Eq. (6.8), is a straightforward metric for how well a model is performing on the task it has been trained on. The most expressive model, ParticleNet-r, has the lowest test loss. PFN-r has the second lowest loss value, and the two light models have very similar test loss. Standard JEC has the highest test loss.

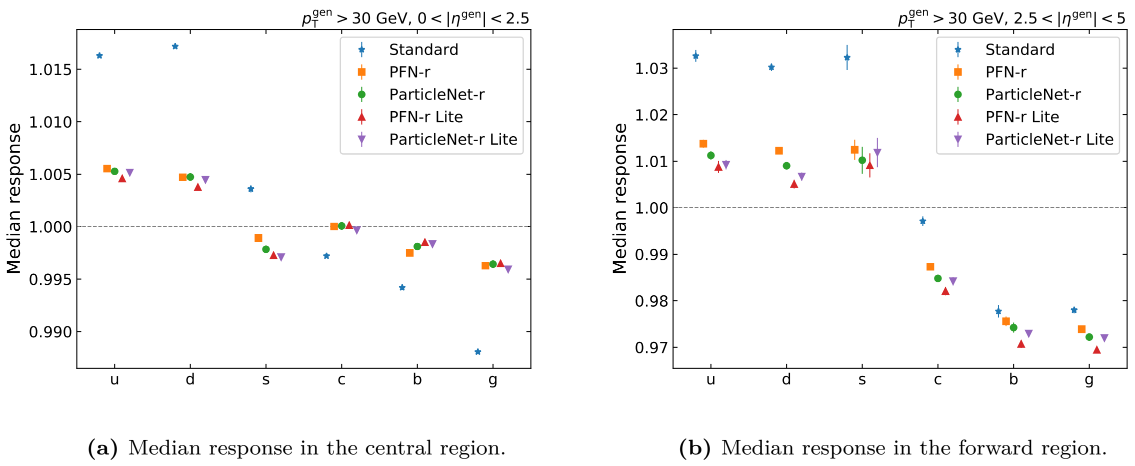
	Parameters	Training step [ms]	Inference time [ms]	Test loss
Standard	**1066**		**0.0005**	0.1143
PFN-r	1 473 313	52	0.0344	0.0789
ParticleNet-r	1 199 553	459	0.4731	**0.0782**
PFN-r Lite	41 305	**50**	0.0344	0.0796
ParticleNet-r Lite	38 417	196	0.1174	0.0799

Table 7.1: Number of parameters, time to train a batch, inference time and test loss for the models. The training step is the time it takes to train on one batch with 1024 elements, and inference time is the time it takes to make predictions for one set of inputs. Both metrics were tested on two Nvidia GeForce RTX 3090 GPUs running in parallel for the neural networks. Standard correction benchmarks were tested on an AMD Ryzen Threadripper 3960X CPU.

The number of parameters of a model in relation to the other metrics can tell us how well a certain architecture is performing. Having a larger PFN-r network seems to affect the computational complexity very little. The time it takes to train one batch with 1024 elements is very similar for both PFN-r networks, and the inference time is identical. More decimals have to be included to notice a difference, but they are as good as indistinguishable from each other. To give an idea, the difference in inference time over the whole test dataset with 2.7 million jets, is 4ms. Since the larger PFN-r network has a noticeable lower loss than the light version and no speed penalty, it can be concluded that more units are purely beneficial when opting for a PFN-r architecture. For ParticleNet-r, that is not the case. The difference in computational complexity for the large model compared to the small one is larger. The training time is more than twice as fast, and the inference time is more than four times as fast for the light version, but still a fair bit slower than PFN-r. The test loss for ParticleNet-r Lite is roughly on par with PFN-r Lite and clearly worse than ParticleNet-r. When accuracy is important the large ParticleNet-r version, with fewer parameters compared to PFN-r, is the preferred model. If inference time is crucial, and PFN-r fails to meet the requirements, then the standard approach is the best option.

7.2 Flavor dependence

The median of the response, Eq. (6.7), for all jet flavors individually is shown in Fig 7.1. All deep learning models successfully reduce the response differences between the jet flavors, or flavor dependence substantially. The reduction is larger in the central region compared to forward region. It can be explained by that the quality of event reconstruction is normally worse in the forward region where no tracking information is available, giving less reliable measurements to train on. The largest improvement in flavor dependence for the response is seen between light quark jets and gluon jets. It is a welcome result since the difference between them is regarded to be a weak point for standard JEC.



(a) Median response in the central region. **(b)** Median response in the forward region.

Figure 7.1: Median energy response for every jet flavor. The improvement over standard JEC is about 70% in the central region and 30% in the forward region of the detector for all models.

The error bars in the plots are produced using the statistical *bootstrapping* technique. The response is randomly sampled 30 times creating new sets of response values equal in size to the original. The median is calculated for each sample individually. The standard deviation is then calculated for those 30 median values, which gives an uncertainty to every point.

The *sum of absolute errors* (SAE) can be used to assess the improvement in flavor dependence. It can be calculated directly from the points in Fig. 7.1 by summing the median response for every flavor subtracted by the mean of those same points and taking the absolute value of the difference. Formally, it can be written as:

$$\mathrm{SAE} = \sum_{\text{flavor}}^{n} \left| R_{50\%,\,\text{flavor}} - \frac{1}{n} \sum_{\text{flavor}}^{n} \left(R_{50\%,\,\text{flavor}} \right) \right| \tag{7.1}$$

where $n = 6$ is the total number of flavors. The relative improvement in flavor dependence for a model compared to standard JEC will be denoted by α, and is defined as:

$$\alpha = 1 - \mathrm{SAE}_{\text{Model}} / \mathrm{SAE}_{\text{Standard}}. \tag{7.2}$$

The values that α can take on ranges from any negative value up to one. A negative result would mean the models perform worse than standard correction, zero is equal to no improvement at all, and $\alpha = 1$ would mean that the median response is the same for all jet flavors.

The flavor dependence results for the central detector region are shown in Table 7.2. Interestingly, PFN-r Lite has the largest improvement, it is 71.85% better compared to standard JEC. PFN-r Lite manages to improve u and d jets more than the other models while still having roughly the same median response as the rest of the models for other jet flavors. The other models hover around 67-68% improvement. PFN-r is particularly good at correcting s jets but slightly worse than other models for b jets. It might seem peculiar that a light model reduces flavor dependence more than its larger counterpart, but it is completely understandable since the regression targets a better jet energy resolution and not a reduction in flavor dependence.

	α [%]	β_{uds} [%]	β_{c} [%]	β_{b} [%]	β_{g} [%]
PFN-r	67.88	11.43	9.69	9.86	7.99
ParticleNet-r	68.11	**12.50**	**10.63**	**10.65**	**8.84**
PFN-r Lite	**71.85**	10.31	8.74	8.78	7.04
ParticleNet-r Lite	67.33	9.32	8.17	8.21	6.55

Table 7.2: Results in the central region of the detector, for $p_{\mathrm{T}}^{\mathrm{gen}} > 30$ GeV jets.

The results for the forward region are shown in Table 7.3. Here too a light model performs the best, this time it is ParticleNet-r Lite with 30.65% improvement over standard JEC. There is more fluctuation in the results between models, but overall the improvements are 28-31% in comparison to standard corrections. The uncertainty in the forward region is larger which stems from the forward region being more noisy in general. Also, the amount of data in the forward region is not as great as in the central region. As seen in Fig. 6.1b, s jets are least represented in the dataset, and their median response uncertainty is also highest. Flavor dependence results in multiple p_{T} bins is available in Appendix A.

	α [%]	β_{uds} [%]	β_{c} [%]	β_{b} [%]	β_{g} [%]
PFN	28.52	14.42	11.35	9.00	9.02
ParticleNet-r	30.26	**14.63**	**11.91**	**9.08**	**9.70**
PFN-Lite	29.20	13.45	10.41	8.26	8.33
ParticleNet-r Lite	**30.65**	13.14	10.81	8.17	9.20

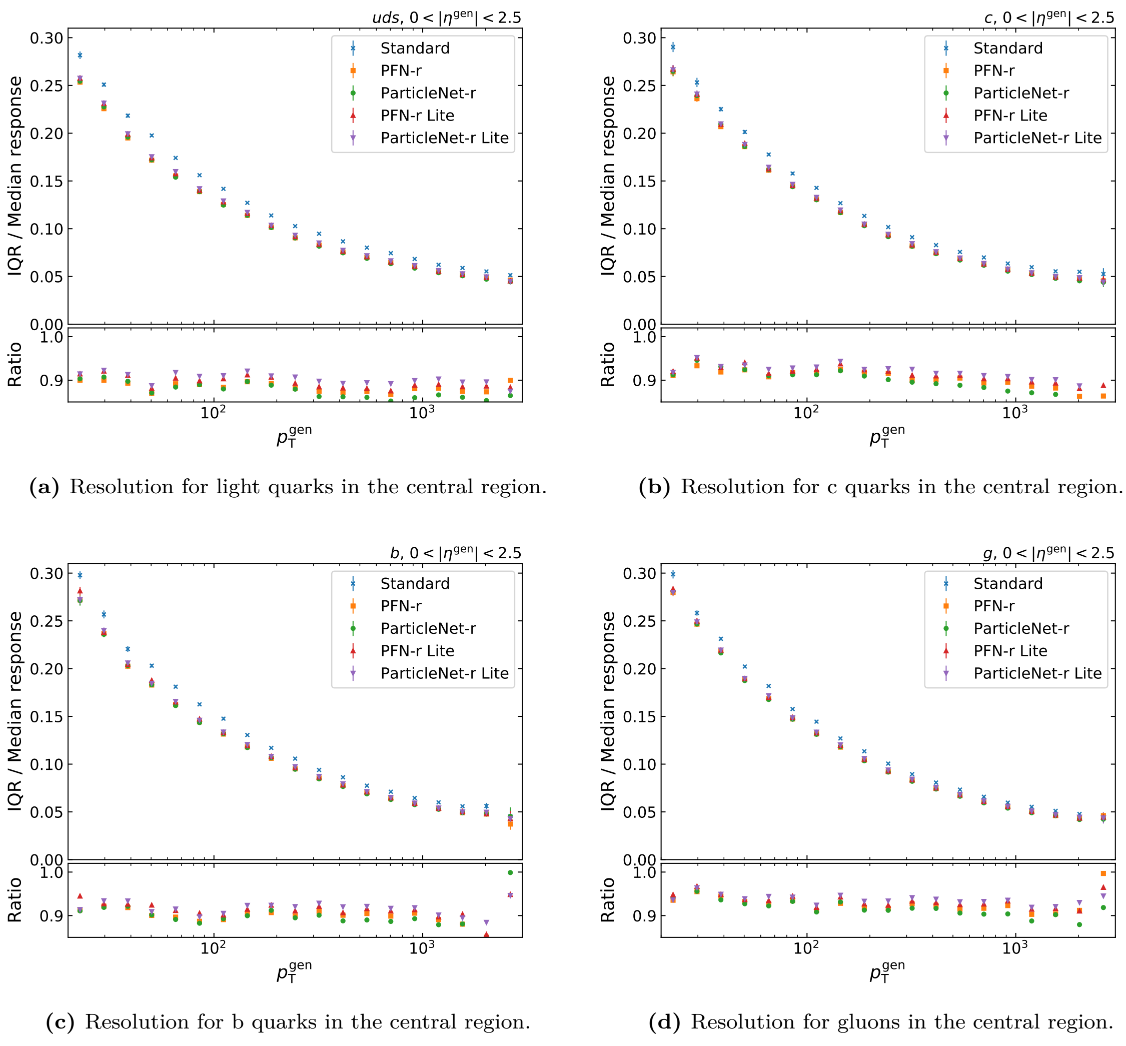
Table 7.3: Results in the forward region of the detector, for $p_{\mathrm{T}}^{\mathrm{gen}} > 30$ GeV jets.

7.3 Jet energy resolution

The regression performance can furthermore be evaluated by looking at the relative jet energy resolution. It is defined here as the *interquartile range* (IQR) divided by the median for the response:

$$\bar{s} = \frac{R_{75\%} - R_{25\%}}{R_{50\%}}. \tag{7.3}$$

IQR can be used as a measure of response resolution, and the median is the middle value separating the greater and lesser halves of the data. Both the median and IQR are robust statistics, meaning that they are less affected by outliers compared to the mean or standard deviation for example. IQR is a robust measure of statistical dispersion, and the median is a robust measure of central tendency.



(a) Resolution for light quarks in the central region.

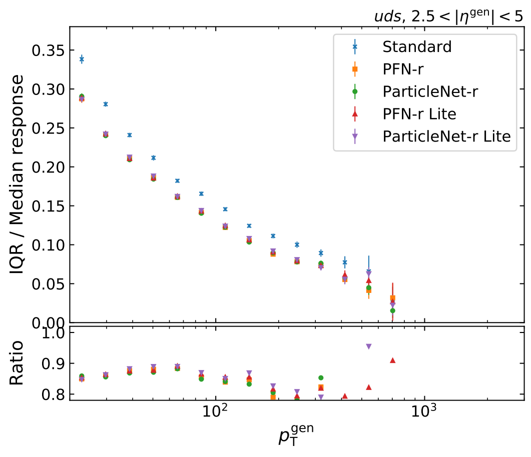
(b) Resolution for c quarks in the central region.

(c) Resolution for b quarks in the central region.

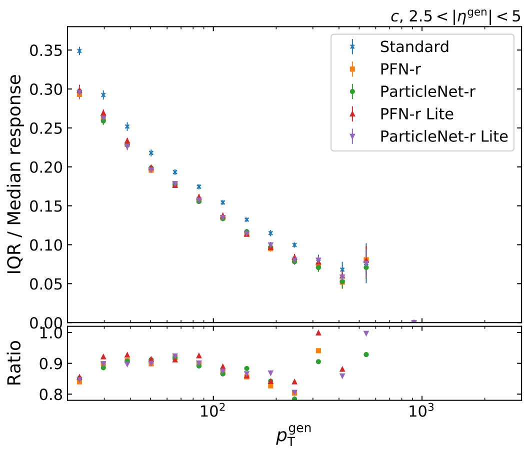
(d) Resolution for gluons in the central region.

Figure 7.2: Resolution for all jet flavors in the central region of the detector. The improvement over standard corrections is on average around 7-12%.

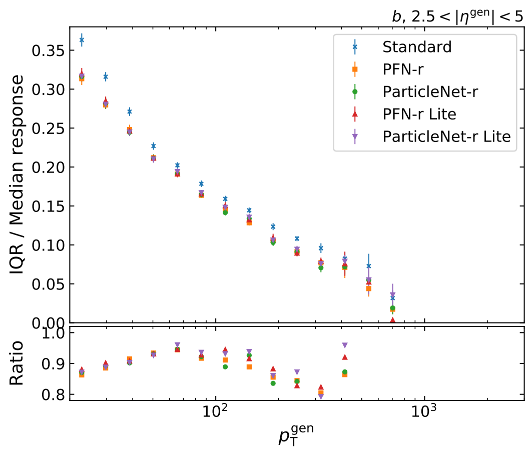
The relative jet energy resolution is shown as a function of generator-level $p_{\mathrm{T}}^{\mathrm{gen}}$ in the upper subplots in both Fig. 7.2 and Fig. 7.3. A smaller value is better. How the resolution compares to standard corrections is shown in the lower subplots as the direct ratio between them. The jets have higher multiplicity and the effect of pileup diminishes at high p_{T}, and therefore the models are expected to perform better at high p_{T}, which they on average do. There are fewer jets in the high p_{T} region and in the high $|\eta|$ region, as Fig. 6.1a show. This causes the bootstrapped uncertainty to be larger there.



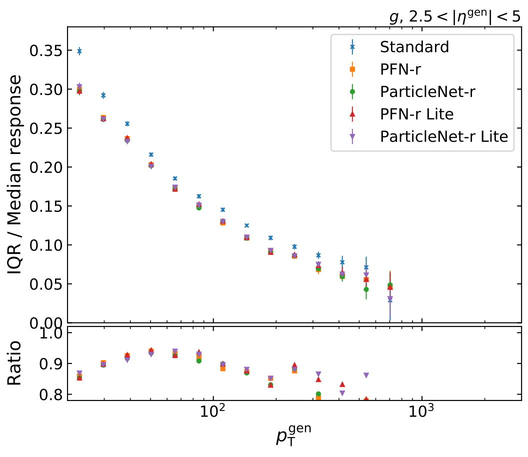
(a) Resolution for light quarks in the forward region.



(b) Resolution for c quarks in the forward region.



(c) Resolution for b quarks in the forward region.



(d) Resolution for gluons in the forward region.

Figure 7.3: Resolution for all jet flavors in the forward region of the detector. The improvement over standard corrections is on average around 12-17%.

The improvement in resolution for a model with respect to standard corrections is denoted by β. It can be calculated from the jet energy resolution ratio:

$$\beta = 1 - \bar{s}_{\mathrm{Model}}/\bar{s}_{\mathrm{Standard}}. \tag{7.4}$$

Table 7.2 show a summary of the resolution improvement in the central region, and Table 7.3 shows the results for the forward region. The models produce a larger improvement in the forward region supposedly because the standard resolution is worse in that region to begin with. Results further divided into low, medium and high p_{T} regions are found in Appendix A. It is clear that the larger models outperform the smaller ones. The resolution improvement also correlates well with the test loss in Table 7.1.

8. Conclusion

The standard set of jet energy corrections are used by almost every physics analysis in CMS, and the requirements on robustness as well as general applicability are particularly high. In this thesis deep learning was used to improve these corrections, bringing the reconstructed transverse momentum closer to the particle-level counterpart. This is possible because jet energy regression can be formalized as a regression problem within the broader context of supervised machine learning.

The neural networks presented in this thesis can make predictions for differently flavored jets. This is in large thanks to the QCD dataset they have been trained on containing equal distributions of light quark jets, gluon jets as well as bottom and charm jets. The data has also been downsampled in overpopulated regions of the parameter space improving the gain in performance for the deep learning models significantly.

Following the latest advancements in deep learning in high energy physics, jets are treated as particle clouds, or unordered sets of particles. It is a very natural way of representing jets that includes all information about the variable length sets of particles contained in the jets. Model architectures able to ingest data in such a format are presented from the recently proposed geometric deep learning framework. The choice of models used in this thesis is grounded on previous studies in jet flavor tagging for which the set-based Particle Flow Network and the graph-based ParticleNet have been developed.

The deep learning models manage to improve the energy resolution and flavor dependence significantly, reaching state of the art performance for jet energy corrections. ParticleNet is observed to have the best overall improvement in results due to the inclusion of locality and a dynamically updating graph. The purely set-based PFN on the other hand is substantially faster to train, and is the preferred architecture if inference time is important. Notably, the standard approach is because of its simplicity much faster still. However, the neural networks yield superior corrections.

Bibliography

[1] The CMS Collaboration, "Jet Energy Calibrations at CMS experiment with 13 TeV collisions," *PoS*, vol. EPS-HEP2017, p. 805, 2017. doi.org/10.22323/1.314.0805.

[2] A. Radovic, M. Williams, D. Rousseau, *et al.*, "Machine learning at the energy and intensity frontiers of particle physics," *Nature*, vol. 560, pp. 41–48, Aug 2018. doi.org/10.1038/s41586-018-0361-2.

[3] D. Guest, K. Cranmer, and D. Whiteson, "Deep learning and its application to lhc physics," *Annual Review of Nuclear and Particle Science*, vol. 68, no. 1, pp. 161–181, 2018. doi.org/10.1146/annurev-nucl-101917-021019.

[4] The CMS Collaboration, "A Deep Neural Network for Simultaneous Estimation of b Jet Energy and Resolution," *Computing and software for big science*, vol. 4, no. 1, p. 10, 2020. doi.org/10.1007/s41781-020-00041-z.

[5] J. Shlomi, P. Battaglia, and J.-R. Vlimant, "Graph neural networks in particle physics," *Machine Learning: Science and Technology*, vol. 2, p. 021001, jan 2021. doi.org/10.1088/2632-2153/abbf9a.

[6] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. Salakhutdinov, and A. Smola, "Deep Sets," 2018. arxiv.org/abs/1703.06114.

[7] H. Qu and L. Gouskos, "Jet tagging via particle clouds," *Physical Review D*, vol. 101, no. 5, 2020. doi.org/10.1103/PhysRevD.101.056019.

[8] M. Thomson, *Modern Particle Physics*. Cambridge University Press, 2013. doi.org/10.1017/CBO9781139525367.

[9] Wikimedia Commons, "Standard model of elementary particles," 2019. w.wiki/49Nk.

[10] L. Evans and P. Bryant, "LHC machine," *Journal of Instrumentation*, vol. 3, no. 08, pp. S08001–S08001, 2008. doi.org/10.1088/1748-0221/3/08/s08001.

[11] J. Gruschke, *Observation of Top Quarks and First Measurement of the $t\bar{t}$-Production Cross Section at a Centre-of-Mass Energy of the 7 TeV with the CMS Experiment at the LHC.* PhD thesis, Karlsruher Institut für Technologie (KIT), 2011. publikationen.bibliothek.kit.edu/1000022394.

[12] The CMS Collaboration, "The CMS experiment at the CERN LHC," *Journal of Instrumentation*, vol. 3, no. 08, pp. S08004–S08004, 2008. doi.org/10.1088/1748-0221/3/08/s08004.

[13] I. Neutelings, "CMS coordinate system," 2017. tikz.net/axis3d_cms.

[14] S. R. Davis, "Interactive Slice of the CMS detector," 2016. cds.cern.ch/record/2205172.

[15] T. Sjöstrand, S. Ask, J. R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, C. O. Rasmussen, and P. Z. Skands, "An introduction to PYTHIA 8.2," *Computer Physics Communications*, vol. 191, pp. 159–177, 2015. doi.org/10.1016/j.cpc.2015.01.024.

[16] M. Bähr *et al.*, "Herwig++ Physics and Manual," *Eur. Phys. J. C*, vol. 58, pp. 639–707, 2008. doi.org/10.1140/epjc/s10052-008-0798-9.

[17] V. Khachatryan, A. M. Sirunyan, A. Tumasyan, W. Adam, E. Asilar, T. Bergauer, J. Brandstetter, E. Brondolin, M. Dragicevic, *et al.*, "Event generator tunes obtained from underlying event and multiparton scattering measurements," *The European Physical Journal C*, vol. 76, no. 3, 2016. doi.org/10.1140/epjc/s10052-016-3988-x.

[18] A. M. Sirunyan, A. Tumasyan, W. Adam, F. Ambrogi, E. Asilar, T. Bergauer, J. Brandstetter, M. Dragicevic, J. Erö, *et al.*, "Extraction and validation of a new set of cms pythia8 tunes from underlying-event measurements," *The European Physical Journal C*, vol. 80, no. 1, 2020. doi.org/10.1140/epjc/s10052-019-7499-4.

[19] P. Nason, "A New Method for Combining NLO QCD with Shower Monte Carlo Algorithms," *Journal of High Energy Physics*, vol. 2004, no. 11, p. 40, 2004. doi.org/10.1088/1126-6708/2004/11/040.

[20] S. Alioli, P. Nason, C. Oleari, and E. Re, "A general framework for implementing NLO calculations in shower Monte Carlo programs: the POWHEG BOX," *Journal of High Energy Physics*, vol. 2010, no. 6, p. 6, 2010. doi.org/10.1007/JHEP06(2010)043.

[21] S. Agostinelli, J. Allison, K. Amako, J. Apostolakis, and H. Araujo, "Geant4 — a simulation toolkit," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 506, no. 3, pp. 250–303, 2003. doi.org/10.1016/S0168-9002(03)01368-8.

[22] B. Webber, "Parton shower Monte Carlo event generators," *Scholarpedia*, vol. 6, no. 12, p. 10662, 2011. Revision #128236, doi.org/10.4249/scholarpedia.10662.

[23] K. Kallonen, "Quantitative Comparison of Deep Neural Networks for Quark/Gluon Jet Discrimination," Master's thesis, University of Helsinki, 2019. helda.helsinki.fi/handle/10138/299781.

[24] The CMS Collaboration, "Particle-flow reconstruction and global event description with the CMS detector," *JINST*, vol. 12, p. P10003. 82 p, 2017. doi.org/10.1088/1748-0221/12/10/P10003.

[25] S. Laurila, *Search for Charged Higgs Bosons Decaying to a Tau Lepton and a Neutrino with the CMS Experiment.* PhD thesis, University of Helsinki, 2019. helda.helsinki.fi/handle/10138/305652.

[26] M. Cacciari, G. P. Salam, and G. Soyez, "The anti-ktjet clustering algorithm," *Journal of High Energy Physics*, vol. 2008, no. 04, pp. 063–063, 2008. doi.org/10.1088/1126-6708/2008/04/063.

[27] H. Kirschenmann, "Jets at CMS and the determination of their energy scale," 2012. phys.org/news/2012-07-jets-cms-energy-scale.html.

[28] The CMS Collaboration, "Jet energy scale and resolution performance with 13 TeV data collected by CMS in 2016-2018," 2020. cds.cern.ch/record/2715872.

[29] Euclid, "Elements," c. 300 BC. farside.ph.utexas.edu/Books/Euclid/Elements.pdf.

[30] F. Klein, "Vergleichende Betrachtungen über neuere geometrische Forschungen," *Mathematische Annalen*, vol. 43, pp. 63–100, 1893. doi.org/10.1007/BF01446615.

[31] M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges," 2021. arxiv.org/abs/2104.13478.

[32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016. deeplearningbook.org.

[33] S. Linnainmaa, "The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors," Master's thesis, University of Helsinki, 1970. ethesis.helsinki.fi/repository/handle/123456789/29618.

[34] P. Veličković, "Theoretical Foundations of Graph Neural Networks," 2021. talks.cam.ac.uk/talk/index/155341.

[35] A. Popov, "Inputs for ML jet calibration," 2020. gitlab.cern.ch/aapopov/ml-jec-vars.

[36] The CMS Collaboration, "CMS Software," 2021. cms-sw.github.io.

[37] G. Petrucciani, A. Rizzi, and C. Vuosalo, "Mini-AOD: A New Analysis Data Format for CMS," *Journal of Physics: Conference Series*, vol. 664, no. 7, p. 072052, 2015. doi.org/10.1088/1742-6596/664/7/072052.

[38] R. Brun and F. Rademakers, "ROOT — An object oriented data analysis framework," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 389, no. 1, pp. 81–86, 1997. doi.org/10.1016/S0168-9002(97)00048-X.

[39] A. Popov, "Choosing training set for DNN jet calibration," 2020. indico.cern.ch/event/924584/#10-pt-spectrum-in-dnn-jet-cali.

[40] "Performance of quark/gluon discrimination in 8 TeV pp data," 2013. cds.cern.ch/record/1599732.

[41] P. Zyla *et al.*, "Review of Particle Physics," *PTEP*, vol. 2020, no. 8, p. 083C01, 2020. doi.org/10.1093/ptep/ptaa104.

[42] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic Graph CNN for Learning on Point Clouds," *ACM Trans. Graph.*, vol. 38, no. 5, 2019. doi.org/10.1145/3326362.

[43] P. T. Komiske, E. M. Metodiev, and J. Thaler, "Energy flow networks: deep sets for particle jets," *Journal of High Energy Physics*, vol. 2019, no. 1, 2019. doi.org/10.1007/JHEP01(2019)121.

[44] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, (Madison, WI, USA), pp. 807–814, Omnipress, 2010. dl.acm.org/doi/10.5555/3104322.3104425.

[45] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (G. Gordon, D. Dunson, and M. DudÃk, eds.), vol. 15 of *Proceedings of Machine Learning Research*, (Fort Lauderdale, FL, USA), pp. 315–323, PMLR, 11–13 Apr 2011. proceedings.mlr.press/v15/glorot11a.html.

[46] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034, 2015. doi.org/10.1109/ICCV.2015.123.

[47] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 448–456, JMLR.org, 2015. dl.acm.org/doi/10.5555/3045118.3045167.

[48] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," 2017. arxiv.org/abs/1412.6980.

[49] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. doi.org/10.1109/CVPR.2016.90.

[50] M. Abadi *et al.*, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015. tensorflow.org.

[51] D. G. Murray, J. Simsa, A. Klimovic, and I. Indyk, "tf.data: A Machine Learning Data Processing Framework," in *Proceedings of the VLDB Endowment*, vol. 14, pp. 2945–2958, 2021. vldb.org/pvldb/vol14/p2945-klimovic.pdf.

[52] Google, "Better performance with the `tf.data` API." tensorflow.org/guide/data_performance, Accessed on Aug. 31, 2021.

[53] L. Gray, N. Smith, A. Novak, *et al.*, "Coffeateam/coffea: Release v0.7.4," June 2021. doi.org/10.5281/zenodo.4901548.

A. p_T binned results

This appendix contains results split up between the central detector region, Table A.1, and the forward detector region, Table A.2. It extends the results presented in Chapter 7 by dividing the flavor dependence α, and resolution β into different p_T intervals. The general trend is that improvements are more pronounced for higher p_T jets.

Interval [GeV]	Model	α [%]	β_uds [%]	β_c [%]	β_b [%]	β_g [%]
$30 < p_\mathrm{T}^\mathrm{gen} < 50$	PFN-r	26.35	**10.95**	**6.90**	**8.43**	6.03
	ParticleNet-r	**26.62**	10.50	5.87	8.13	**6.24**
	PFN-r Lite	25.05	9.61	5.70	7.14	5.17
	ParticleNet-r Lite	21.31	9.15	5.62	7.28	4.68
$50 < p_\mathrm{T}^\mathrm{gen} < 100$	PFN-r	41.17	11.29	8.40	10.99	7.31
	ParticleNet-r	38.86	**11.78**	**8.68**	**11.37**	**7.36**
	PFN-r Lite	**41.48**	10.22	7.60	9.60	6.41
	ParticleNet-r Lite	36.90	9.14	7.22	9.56	5.93
$100 < p_\mathrm{T}^\mathrm{gen} < 300$	PFN-r	57.09	11.17	8.10	9.84	8.05
	ParticleNet-r	57.29	**11.69**	**8.80**	**10.10**	**8.22**
	PFN-r Lite	**60.44**	9.81	7.20	8.52	6.99
	ParticleNet-r Lite	55.49	8.98	6.89	8.04	6.31
$300 < p_\mathrm{T}^\mathrm{gen} < 1000$	PFN-r	72.37	12.55	10.12	9.66	8.06
	ParticleNet-r	73.32	**13.98**	**11.33**	**10.77**	**9.04**
	PFN-r Lite	**76.68**	11.68	9.30	8.55	7.20
	ParticleNet-r Lite	73.56	10.47	8.44	7.90	6.56
$p_\mathrm{T}^\mathrm{gen} > 1000$	PFN-r	81.36	11.73	11.46	10.70	9.30
	ParticleNet-r	84.43	**13.19**	**13.30**	**11.87**	**10.40**
	PFN-r Lite	**87.20**	10.76	10.62	9.86	8.18
	ParticleNet-r Lite	84.16	9.59	9.84	9.36	7.82

Table A.1: Summary of jet energy regression results in the $0 < |\eta^\mathrm{gen}| < 2.5$ region.

Interval [GeV]	Model	α [%]	β_{uds} [%]	β_{c} [%]	β_{b} [%]	β_{g} [%]
$30 < p_{\mathrm{T}}^{\mathrm{gen}} < 50$	PFN-r	7.51	12.79	9.97	9.09	8.27
	ParticleNet-r	10.81	**13.66**	10.16	9.52	8.50
	PFN-r Lite	10.33	12.47	8.61	8.90	8.08
	ParticleNet-r Lite	**12.27**	12.53	**10.36**	**9.79**	**9.19**
$50 < p_{\mathrm{T}}^{\mathrm{gen}} < 100$	PFN-r	27.87	12.43	9.98	7.36	7.42
	ParticleNet-r	27.34	**12.62**	**10.20**	**7.44**	**8.17**
	PFN-r Lite	26.26	11.86	9.25	7.12	6.55
	ParticleNet-r Lite	**28.41**	11.33	9.33	6.54	6.94
$100 < p_{\mathrm{T}}^{\mathrm{gen}} < 300$	PFN-r	48.59	18.12	14.04	10.93	12.41
	ParticleNet-r	48.03	**18.55**	**14.49**	**11.33**	**13.15**
	PFN-r Lite	52.51	16.47	13.18	9.83	12.04
	ParticleNet-r Lite	**52.36**	16.41	13.33	8.71	11.72
$300 < p_{\mathrm{T}}^{\mathrm{gen}} < 1000$	PFN-r	61.26	24.54	**25.55**	**18.45**	**24.12**
	ParticleNet-r	65.43	19.98	20.30	13.95	23.74
	PFN-r Lite	62.10	18.66	20.48	10.39	22.82
	ParticleNet-r Lite	**68.02**	**26.34**	16.65	14.71	19.10

Table A.2: Summary of jet energy regression results in the $2.5 < |\eta^{\mathrm{gen}}| < 5$ region.