MARMARA UNIVERSITY

INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

Machine Learning Techniques for JetMET Data Certification of the CMS Detector at CERN

Jawaher ALTORK

MASTER THESIS

Department of Physics

Thesis Supervisor

Prof. Dr. Mithat KAYA

Thesis Co-Supervisor

Dr. Ismail Okan ATAKISI

ISTANBUL, 2023

MARMARA ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

CERN'de CMS Dedektörünün JetMET Veri Sertifikasyonu için Makine Öğrenimi Teknikleri

Jawaher ALTORK

YÜKSEK LİSANS TEZİ

Fizik Bölümü

DANIŞMAN

Prof. Dr. Mithat KAYA

EŞ-DANIŞMAN

Dr. İsmail Okan ATAKİŞİ

İSTANBUL, 2023

Acknowledgements

I would like to express my heartfelt gratitude to the following individuals and organizations for their invaluable support and contributions throughout the completion of my master's thesis.

I would like to express my sincere gratitude to my advisor, Prof. Mithat Kaya, and my co-advisor, Dr. I. Okan Atakisi, for their invaluable guidance and support throughout my master’s program. Their knowledge, encouragement, and mentorship have played a pivotal role in shaping this thesis.

I extend my deepest appreciation to my parents for their unwavering love, encouragement, and constant belief in my abilities. Their support has been the pillar of strength throughout my master’s journey. I am also grateful to my brother and my friends for their continuous support, understanding, and encouragement. Their presence has been a constant source of motivation and joy, making this journey memorable and enjoyable.

I would also like to acknowledge and express my gratitude to CERN for providing me with the opportunity to work with its exceptional research infrastructure. Their commitment to scientific excellence and advancement has been truly inspiring.

Lastly, I would like to extend my thanks to YTB Turkiye Burslari for granting me the scholarship that has made my educational pursuit possible. Their support has not only alleviated financial burdens but has also provided me with the opportunity to immerse myself in an enriching academic environment.

I am deeply grateful to all those mentioned above for their immense contributions, guidance, and support, without which this thesis would not have been possible.

Abstract

For cutting-edge detectors like the compact muon solenoid (CMS) detector, where high energetic particles emerging from proton-proton collisions at the CERN large hadron collider (LHC) are measured, data quality monitoring (DQM) and data certification (DC) are crucial components in ensuring reliable data quality suitable for physics analysis. In the offline DQM procedure, the quality of recorded data, grouped in 'runs', is evaluated. The current method for certification of quantities related to hadronic jets and missing transverse momentum (MET) is mostly reliant on manually monitoring reference histograms summarizing the status and performance of the detector. Given the large number of distributions that are mentioned, the process is time intensive and prone to human error when deviations from the norm are less evident. The results presented here show machine learning methods for certifying offline DQM data, focusing on hadronic jet and MET objects. Using collision data collected during 2018, we show that autoencoder techniques can accurately certify runs and detect ineffective detector regions, allowing us to reduce both the time required for DC, as well as the rate of anomalies a human expert can potentially miss.

ÖZET

CERN Büyük Hadron Çarpıştırıcısında proton-proton çarpışmalarından çıkan yüksek enerjili parçacıkların ölçüldüğü kompakt müon solenoid (CMS) dedektörü gibi son teknolojiyi kullanan dedektörler için, veri kalitesi izleme (DQM) ve veri sertifikasyonu (DC) yapmak, fizik analizleri ve veri kalitesini sağlamak için önemli iki bileşendir. Çevrimdışı DQM işlemlerinde, veri kümesi (Run) olarak gruplanan kayıtlı verilerin kalitesi değerlendirilir. Hadronik jetler ve eksik enine momentum (MET) ile ilgili miktarların sertifikasyonu için mevcut yöntem, çoğunlukla dedektörün durumunu ve performansını özetleyen referans histogramlarının manuel olarak izlenmesine dayanır. Bahsedilen çok sayıda histogram göz önüne alındığında, süreç zaman alıcıdır ve normdan sapmalar daha az belirgin olduğunda insan hatasına eğilimlidir. Burada sunulan sonuçlar, hadronik jet ve MET nesnelerine odaklanarak çevrimdışı DQM verilerini iyi veya kötü veri olarak onaylamak için makine öğrenimi yöntemlerini göstermektedir. Bu tez de 2018 yılı boyunca toplanan çarpışma verilerini kullanarak, “autoencoder” tekniklerinin çalışmaları doğru bir şekilde onaylayabildiğini ve düzensiz dedektör bölgelerini tespit edebildiğini, böylece hem veri sertifikasyonu için gereken süreyi hem de bir uzman kişinin potansiyel olarak gözden kaçırabileceği anormallik oranını azaltabildiğini gösterdik.

Contents

1. Introduction **4**

2. Theoretical Background **5**

2.1. The Standard Model of Particle Physics 6

2.2. Overview of the LHC 8

2.3. The CMS Detector 11

2.4. Object Reconstruction in CMS 15

2.4.1. Particle-Flow Algorithm 15

2.4.2. Jet and MET 17

2.5. The CMS DQM and DC 20

3. Machine Learning Algorithms **27**

3.1. Supervised Learning 28

3.1.1. K- Nearest Neighbours 29

3.1.2. Gaussian Naïve Base 31

3.1.3. Support Vector Machine 33

3.2. Neural Networks 36

3.3 Autoencoders 44

4. Analysis **47**

4.1. Data Sets Description 47

4.2. Methodology 51

5. Results **51**

5.1. Supervised Learning 51

5.1.1. Choice of Input Features: Feature Reduction 52

5.1.2. DC with Classification Algorithms 56

5.2. Unsupervised Learning 60

5.2.1. DC with Autoencoders 61

5.2.2. Anomaly Detection with Autoencoders: MET Histogram. 69

5.2.2. Anomaly Detection with Autoencoders: 2-d Histograms 74

Summary and Conclusion **84**

References **85**

ABBREVIATIONS/ SYMBOLS

SM **:** Standard Model of Particle Physics

BSM **:** Beyond the Standard Model

QCD **:** Quantum Chromodynamics

EW **:** Electroweak

LHC **:** Large Hadron Collider

DQM **:** Data Quality Monitoring

DC **:** Data Certification

MET **:** Missing Transverse Energy

CMS **:** Compact Muon Solenoid

DAQ **:** Data Acquisition System

ECAL **:** Electromagnetic Calorimeter (ECAL)

HCAL **:** Hadronic Calorimeter (HCAL)

HB **:** HCAL Barrel detector

HE **:** HCAL Endcap detector

HF **:** HCAL Forward detector

HO **:** HCAL Outer detector

EB **:** ECAL Barrel detector

EE **:** ECAL Endcap detector

HLT **:** High-Level Trigger

CHS **:** Charged Hadron Subtraction

MHz : Mega Hertz

KHz : Kilo Hertz

ns : Nano Second

T : Tesla

GeV : Giga Electron Volt

R : Distance

b : Barn

$\boldsymbol{\phi}$: Azimuth Angle

$\boldsymbol{\eta}$: Pseudorapidity

$\boldsymbol{P_T}$: Transverse Momentum

$\boldsymbol{E_T}$: Transverse Energy

$\boldsymbol{\sigma}$: Standard Deviation

$\boldsymbol{\mu}$: Mean

PF : Particle Flow

IP : Interaction Point

METSig : MET Significance

LS : Luminosity Section

DPG : Detector Performance Group

POG : Physics Object Group

PAG : Physics Analysis Group

JME : Jet and MET

CMSSW : CMS Software

GUIs : Graphical User Interfaces

ML : Machine Learning

AI : Artificial Intelligence

CHF : Charged Hadron Energy Fraction

KNN : K-Nearest Neighbors

GNB : Gaussian Naive Bayes

SVM : Support Vector Machine

MMH : Maximum Marginal Hyperplane

RBF : Radial Basis Function

ANNs : Artificial Neural Networks

FCCNs : Fully Connected Neural Networks

MSE : Mean Squared Error

MAE : Mean Absolute Error

AUC-ROC : Area Under the Receiver Operating Characteristic Curve

TPR : True Positive Rate

FPR : False Positive Rate

AE : Autoencoder

MLP : Multi-Layer Perceptron

1. Introduction

The Standard Model (SM) of particle physics is the ultimate cumulation of many theories and experimental results aiming to describe the elementary particles constructing the universe around us at its most fundamental level, and to define the interactions perceived by these tiny building blocks. Despite being one of the greatest achievements in modern physics, the SM, like any other model, has some limitations and shortcomings, for this reason, many approaches beyond the SM attempted to overcome these flaws.

Hence, particle colliders like the Large Hadron Collider (LHC) at CERN are made to examine the current state of theory, search for unexplored phenomena, and identify particles not yet encompassed within the existing framework. The precise measurement of the parameters defined by the SM plays a critical role in both endeavors. These measurements serve the purpose of testing theoretical predictions and gaining a deeper understanding of background processes and the final outcomes of analyses. So, an integral element of the LHC analyses entails the reconstruction of physical objects such as electrons, muons, and jets. In both the pursuit of high-precision tests on the SM and the quest for new particles, monitoring of the collected data plays a vital role in ensuring optimal efficiency of the detector and its operations, as well as in guaranteeing the reliability of the data for physics analyses.

The Data Quality Monitoring (DQM) system implemented by the CMS detector at CERN has the objective of identifying any anomalies in data reconstruction or calibration. It plays a crucial role in certifying the data for selection in physics analyses. The monitoring process involves the use of histograms that store information derived from both raw and reconstructed data obtained from individual detectors. Specifically, for offline DQM, focusing on quantities associated with hadronic jets and missing transverse momentum (MET), the Data Certification (DC) process is primarily done manually by DQM shifters who compare the histograms to reference standards. However, this human-centric approach of DQM has certain drawbacks, including the consumption of time and manpower and susceptibility to human errors. This dissertation uses different machine learning models as a tool to automate the offline data certification process for Jet and MET related observables, leaving only non-trivial cases for experts to certify.

This work involves the development of novel unsupervised algorithms to certify runs of Jet and MET plots. Additionally, the study aims to detect anomalous regions within the detector by utilizing various JetMET histograms. Notably, the application of unsupervised learning to offline JetMET DC represents a novel approach that has not been previously explored. This approach offers significant advantages, including the ability to train the algorithms directly with data in a short timeframe while maintaining high accuracy levels.

The structure of this work is organized as follows: Chapter 2 provides a theoretical background, focusing on key elements such as the standard model, the large hadron collider, the CMS detector, the particle-flow algorithm, and the Jet and MET objects. Chapter 3 delves into both supervised and unsupervised machine learning models employed in this study, discussing their characteristics and applications. Furthermore, Chapter 4 outlines the utilized datasets and the methodology employed for analysis. In Chapter 5, the obtained results are presented and thoroughly discussed. This section critically evaluates the approaches, methods, and assesses the validity and reliability of the results. Finally, Chapter 6 serves as a summary of the entire work and offers prospects for future research and expansion in this field.

2. Theoretical Background

Particle physics seeks to unravel the mysteries of the observable universe by deconstructing it into its fundamental constituents and examining the interactions that govern their behavior. The origins of this field can be traced back to the advent of quantum mechanics during the early 20th century, which laid the foundation for a comprehensive understanding of the universe's observable state. At the core of this investigation lies the standard model of elementary particles, which stands as the most rigorously tested and validated theory in physics. Serving as the cornerstone of particle physics, the SM provides a framework for addressing numerous questions regarding the observable state of the universe.

This chapter will describe the basics of the SM theory, followed by an explanation of the LHC and its detectors. The focus will be on the CMS detector, which plays a vital role in the analysis of this thesis. Additionally, the CMS DQM and DC systems will be explained.

2.1. The Standard Model of Particle Physics

The current best interpretation of the physical world at the subatomic level is the standard model of particle physics, describing the constituents of matter and antimatter as well as their interactions.

In the SM, there are twelve fundamental particles with half-integer spin, called fermions, whose interactions are mediated by the strong, weak, and electromagnetic (EM) forces, as summarized in Fig. 2.1. Quarks are the building blocks of protons and neutrons, which make up the atomic nucleus. Leptons consist of electrons, muon, tau, and their corresponding neutrinos. Electrons are found in atoms and are involved in chemical bonding, while muons and taus are heavier versions of the electron. Neutrinos are electrically neutral and are difficult to detect because they interact only weakly with matter. Each fermion in the SM has an associated antiparticle, possessing an identical mass but opposite charge. The forces in the SM are mediated by "force carrier" particles known as gauge bosons, characterized by their integer spin. Quantum Chromodynamics (QCD) and the Electroweak (EW) sectors of the SM determine the properties and interactions of these particles. The strong force, mediated by the exchange of gluons, binds quarks together. The weak force, mediated by the exchange of W and Z bosons, plays a role in various nuclear processes. The electromagnetic force, governed by the exchange of photons, includes interactions between electrically charged particles. The final particle described by the SM is the Higgs Boson, characterized by a spin of zero. Its discovery, achieved by the ATLAS and CMS experiments in 2012 [1], utilized data collected during the initial operation period of the LHC accelerator, referred to as Run I.

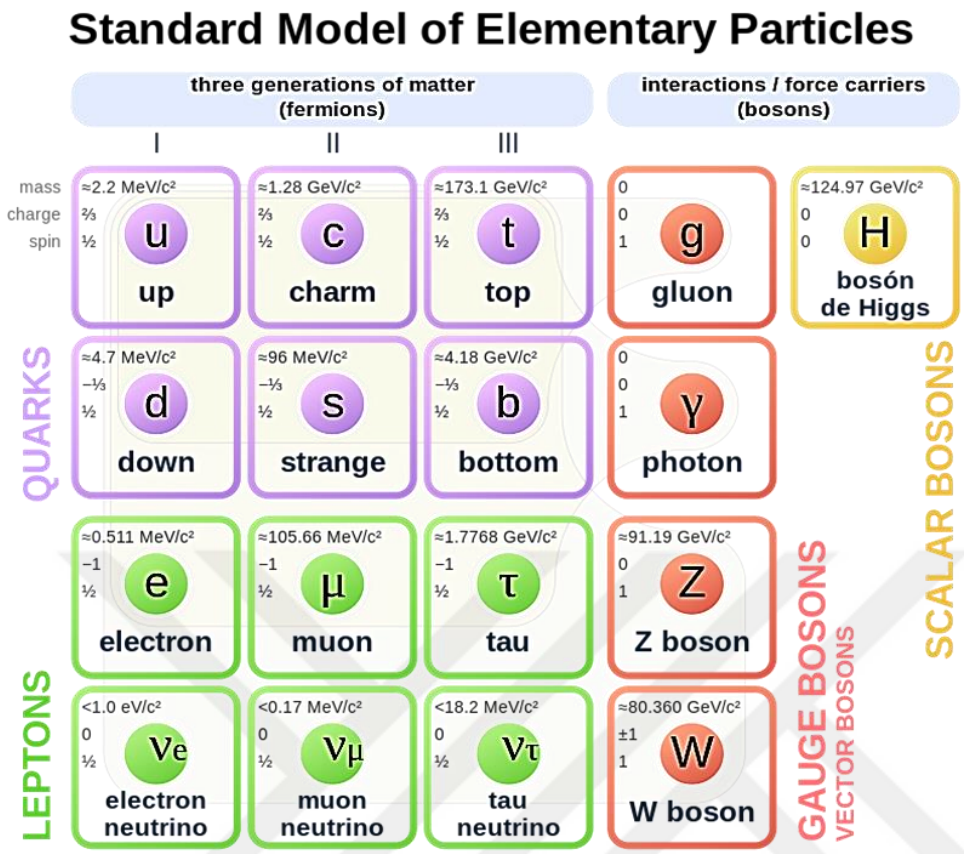


Figure 2.1. Visual representation of the particle composition within the Standard Model, organized into groups of bosons and fermions across three generations [2].

Developed in the early 1970s, the SM has undergone extensive testing and validation through particle physics experiments, yet it remains an incomplete theory. While successful in describing and predicting various phenomena, the model lacks a unified framework encompassing all fundamental forces. In particular, constructing a theory of gravity that parallels those for other forces poses significant challenges. Moreover, the SM falls short in explaining fundamental mysteries, such as the prevalence of matter over antimatter in the universe and the existence of dark matter observed on cosmological scales. It also fails to account for the accelerated expansion of the universe (attributed to dark energy) and the disproportionately high mass of galaxies compared to what is accounted for by visible matter alone (with dark matter as a prominent candidate). These represent only a selection of unanswered questions that lie beyond the scope of the SM.

Particle physics experiments play a crucial role in examining and validating models of the universe. These experiments serve multiple purposes, including confirming predictions made by the SM, investigating new physics processes proposed by Beyond Standard Model

(BSM) theories, and searching for unknown phenomena lacking theoretical explanations. Specifically, experiments focused on exploring BSM physics enable us to gain insights into phenomena like dark matter, dark energy, and the properties of the Higgs boson, among other intriguing aspects. The following section provides an overview of the world's most potent particle accelerator and its associated detectors.

2.2. Overview of the LHC

Located near Geneva, Switzerland, the Large Hadron Collider stands as the largest and most powerful particle accelerator worldwide. More details of all aspects of the LHC can be found in [3].

As the name implies, the LHC is a massive particle accelerator with a circumference of around 27 kilometers. Inside the accelerator, two beams of particles of the same kind, either protons or lead ions, travel at close to the speed of light in opposite directions before they are made to collide. Within the LHC, the beams are deliberately brought into collision at four specific points along the accelerator ring, aligning precisely with the locations of four particle detectors: ATLAS, CMS, ALICE, and LHCb. The ATLAS [4] and CMS [5] experiments are designed to search for new particles predicted by the SM and by theories beyond. This thesis centres around the CMS experiment, which will be further explored and discussed in detail in the next section of this chapter. On the other hand, LHCb [6] places its emphasis on comprehending the matter-antimatter asymmetry that existed in the early universe by studying the decays of b-hadrons. Lastly, ALICE [7] focuses on investigating the properties of strongly interacting matter formed during collisions of heavy ions.

The LHC employs a vacuum tube to circulate particles, which are manipulated using electromagnetic devices. To ensure precise beam size and position, a multitude of magnets, including superconducting dipole magnets for maintaining circular particle orbits, quadrupole magnets for beam focusing, and magnets for "squeezing" particles closer together to enhance collision probabilities, are utilized. The LHC's accelerating cavities function as electromagnetic resonators, not only accelerating particles but also compensating for energy losses to maintain a constant energy level, thereby maximizing collision rates. With its powerful magnet design, the LHC achieves proton acceleration up to 7 TeV per proton, enabling high-energy proton-proton (pp) collisions with a centre-of-

mass energy of up to $\sqrt{s}$ = 14 TeV. Moreover, these electromagnets are constructed using superconducting coils, enabling efficient electricity conduction without resistance or energy loss.

Inside the accelerator, the protons are arranged into bunches, with each bunch containing millions of protons. These bunches are separated from each other by a small distance within the beam pipe. The LHC operates by colliding two bunches of protons at a time, with each bunch traveling in opposite directions around the LHC ring. The protons in each bunch are made to collide with each other at very high energies, producing a shower of subatomic particles that are then detected by large particle detectors located around the LHC ring.

The number of proton bunches in the LHC can vary depending on the operating conditions, but typically there are several hundred bunches in each beam. The number of bunches in each beam and their separation in time determine the LHC's luminosity (L) [8]. Luminosity is a measure of the rate at which proton-proton collisions occur. It is the measurement of the number of collisions that can be produced in a detector per $\mathrm{cm}^2$and per second. The higher the luminosity, the more collisions occur, and the more data can be collected for particle physics research. For head-on collisions, which generate the highest possible energy collisions, the luminosity can be calculated as:

$$L = \frac{N_1 N_2 n_p f}{4\pi\sigma_x\sigma_y} \tag{2.1}$$

The luminosity of a particle collider is determined by various factors, including the number of particles in each bunch (N), the frequency at which the particles complete turns around the ring (f), the number of bunches (n_p), and the beam cross-section (σ). The cross section refers to a measure of the effective area of interaction between the particles and is typically expressed in units of barn ($b = 10^{-24}\, cm^2$).

In brief, to achieve higher luminosity at the Large Hadron Collider, it is necessary to increase the number of protons in each bunch and to decrease the size of the bunches. This allows more particles to be squeezed into a smaller space around the interaction region, increasing the chances of proton-proton collisions and thereby increasing the rate of data collection.

The integral of the delivered luminosity over time is called integrated luminosity. It is calculated by integrating the instantaneous luminosity (the luminosity at a given moment) over time. Integrated luminosity is an important value to characterize the performance of the accelerator as it determines the number of proton-proton collisions that can be expected during that period, which in turn affects the ability to detect rare or highly energetic particles. The unit of integrated luminosity is typically expressed in inverse femtobarns (fb^{-1}), where a femtobarn is a unit of area used to measure the cross-sectional area of particle interactions. One inverse femtobarn ($1\ fb^{-1}$) corresponds to approximately 70×10^{12} proton-proton collisions.

In order to achieve the desired peak instantaneous luminosity of $L = 10^{34}\ cm^{-2}\ s^{-1}$, the LHC is designed to circulate 2808 bunches for each proton beam, with each bunch containing 10^{11} protons. The paths of these bunches intersect every 25 ns, referred to as the bunch crossing interval, at the collision points. Within each crossing, there can be up to 60 pp interactions, leading to an astounding rate of approximately 1 billion collisions per second.

The high rate of particle collisions can lead to what is known as pile-up. Pile-up arises when multiple pp collisions occur within the same bunch crossing, producing additional particles that can interfere with the reconstruction of the primary interaction. This causes many challenges such as deadtime and buffer flow. Deadtime refers to the time during which data cannot be recorded due to processing of previous events, while buffer flow refers to the capacity of the data acquisition systems to handle the data being produced. This can lead to inefficiencies in data collection and a reduction in the overall data quality. This may happen due to the high luminosity of the LHC and the fact that proton bunches are not perfectly synchronized, so some collisions occur slightly before or after the primary interaction.

The underlying event is another unwanted activity that may occur in the collision. It refers to the additional particles produced in proton-proton collisions, beyond the hard scattering process of interest. These additional particles arise from the exchange of gluons and can mask the signal of the process being studied.

The trigger and Data Acquisition (DAQ) systems of the LHC [9] are designed to handle the high data rates and reduce the effects of pile-up and underlying events. The trigger system is responsible for deciding which events are interesting enough to be recorded for further

analysis. It does this by looking for specific signatures or patterns of particles that may indicate the presence of interesting physics. The trigger system operates through multiple levels, progressively increasing selectivity and effectively reducing the rate of events that require further processing. The data acquisition system then takes the data from the interesting events and stores it for further analysis. However, the data acquisition system can only record a limited number of events per second, so it's important to ensure that the trigger system is efficient at selecting the most interesting events. To mitigate the effects of pile-up and underlying events, the two systems are designed to read out a specific time window around each collision event to mitigate the effects of pile-up and underlying events. This time window is chosen based on the expected duration of the collision, which is typically a few nanoseconds. By only recording data within this time window, the signal from each collision event can be separated from other events and the effects of overlapping particles can be reduced.

The LHC has successfully completed two operational periods to date. The first period, known as Run-I, spanned from 2009 to 2012, during which pp collisions took place at energy levels of 7 and 8 TeV. This was followed by Run-II, which lasted from 2015 to 2018 and featured pp collisions at an increased energy of 13 TeV. Currently, the LHC has commenced its third operational period, known as Run-III, which is scheduled to run from 2022 to 2025. During Run-III, the proton beams collide at a centre-of-mass energy of 13.6 TeV, enabling enhanced precision and discovery potential. The primary objectives of Run-III include accumulating more data to study the properties of the Higgs boson and conducting searches for physics beyond the Standard Model. For this study, we primarily focus on working with data from the 2018 Run-II release. Additionally, we also analyse data obtained from the new 2022 Run-III release.

2.3. The CMS Detector

The Compact Muon Solenoid serves as a versatile detector at the LHC, fulfilling multiple objectives. Its primary purpose is to accurately determine the momentum and energy of stable particles resulting from proton-proton collisions. It is composed of several subsystems, including trackers, calorimeters, a solenoid, and a muon system, which work together to reconstruct and identify various particles produced in high-energy collisions as shown in Fig. 2.2. The CMS detector is designed to detect a wide range of particles,

including photons, electrons, muons, and hadrons, making it a valuable tool for studying the properties of the SM particles and searching for new physics beyond the SM.

At the heart of the CMS apparatus lies a superconducting solenoid with a diameter of 6 m, generating a magnetic field of approximately 4 T. This solenoid plays a critical role in providing the magnetic field necessary for the operation of the CMS detector's sub-detectors, which encompass the silicon pixel and strip tracker, the crystal Electromagnetic Calorimeter (ECAL), and the brass-scintillator Hadronic Calorimeter (HCAL). These sub-detectors are positioned within the volume of the solenoid's magnetic field. Outside of the solenoid volume, there are muon sub-detectors, positioned within a flux-return yoke.

The HCAL consists of four subdetectors, a Barrel Detector (HB), which provides coverage in the pseudorapidity range $|\eta| < 1.3$; two Endcap Detectors (HE), which cover the range $1.3 < |\eta| < 3.0$; two forward detectors (HF), which cover the range $2.8 < |\eta| < 5.0$, and a detector outside of the solenoid (HO), which covers the range $|\eta| < 1.3$. In contrast, the ECAL covers the pseudorapidity range $|\eta| < 1.479$ in the barrel region (EB) and the range $1.479 < |\eta| < 3.0$ in two endcap regions (EE) [10][5].

At the LHC, proton bunches travel in opposite directions and collide at a frequency of 40 MHz with a time spacing of 25 ns. This means that proton bunches are designed such that they cross each other's paths every 25 ns. However, due to the storage and computing limitations, only a small fraction of this data can be recorded and analyzed. To overcome this challenge, the CMS trigger system is employed.

The CMS employs a two-tiered trigger system to select events of interest for further analysis. The first level of the trigger system called the Level-1 trigger [10], is hardware-based and employs custom-designed electronics to analyze data from the CMS detector in real-time. The Level-1 trigger reduces the data rate from the detector from 40 MHz to around 100 kHz by identifying events that contain high-energy particles such as electrons, muons, photons, and jets. The second level of the trigger system called the High-Level Trigger (HLT) [11], is software-based and runs on a large cluster of computers. The HLT performs a more detailed analysis of the data selected by the Level-1 trigger and applies complex algorithms to further reduce the data rate to around 1 kHz. These events, which have passed both levels of the trigger system, are then saved to disk for offline analysis by physicists. This analysis involves studying the data in detail, using specialized software and

computing resources to search for interesting patterns or anomalies that could provide new insights into the fundamental nature of matter and the universe.

Triggers are typically based on the identification of a certain number of trigger objects: muons are identified by their ability to travel through many interaction lengths of material without interacting, making them distinguishable from other particles; photons and electrons are identified by their ability to be absorbed more easily than hadrons; Jets, which are produced by quarks and gluons, have both electromagnetic and hadronic content and are identified based on their distinctive signatures in the detector.

The CMS adopts a specific coordinate system, with the origin positioned at the nominal collision point (IP) within the detector (refer to Fig. 2.3). In this system, the y-axis points vertically upwards, the x-axis points radially inwards towards the center of the LHC, and the z-axis aligns with the direction of the beam. The azimuthal angle φ is measured from the x-axis within the x-y plane, while the polar angle θ is measured from the z-axis.

Due to the high energies of the colliding particles, there is a likely boost in the z-direction. This makes the polar angle θ a less useful variable for measuring distances, as it can be affected by longitudinal Lorentz boosts. To address this issue, the variable pseudorapidity (η) is constructed, whose differences are Lorentz invariant under longitudinal Lorentz boosts,

$$\eta = -ln\left[tan\left(\frac{\theta}{2}\right)\right] \tag{2.2}$$

The distance ΔR between two objects i and j is calculated using the differences $\Delta\varphi = \varphi_i - \varphi_j$ and $\Delta\eta = \eta_i - \eta_j$,

$$\Delta R = \sqrt{\Delta\varphi^2 + \Delta\eta^2} \tag{2.3}$$

The transverse momentum (p_T) and transverse energy (E_T) are calculated from the x and y components, while the imbalance of energy measured in the transverse plane is referred to as E_T^{miss}.

For momentum measurements, the use of a z-component is again impractical. Instead, the transverse momentum (p_T) is employed, representing the momentum component perpendicular to the beam axis. The transverse momentum is derived from the x and y-

components (p_x and p_y) of the object's total momentum, such as a particle or jet. Its magnitude is calculated as follows:

$$p_T = \sqrt{p_x^2 + p_y^2} \tag{2.4}$$

These four variables (E, p_T, η, ϕ) provide a complete description of the particle's momentum and mass in the CMS coordinates. Further information on the CMS detector, including more details of the coordinate system and relevant kinematic variables, can be found in [5].

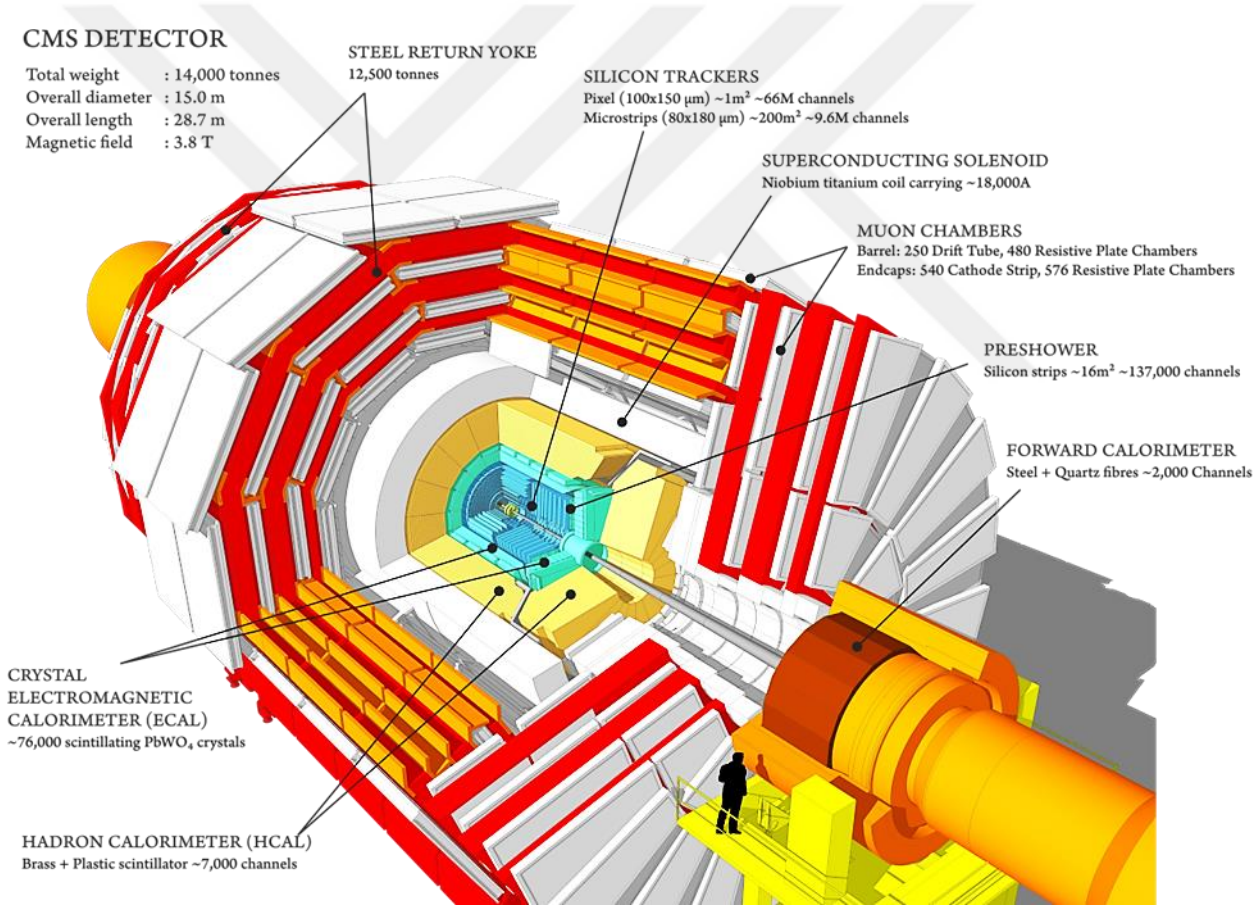


Figure 2.2. Sketch of the CMS detector and its subsystems. The table in the upper left corner provides an overview of the basic properties, including weight and dimensions. Each component of the detector is labelled and depicted in distinct colours [12].

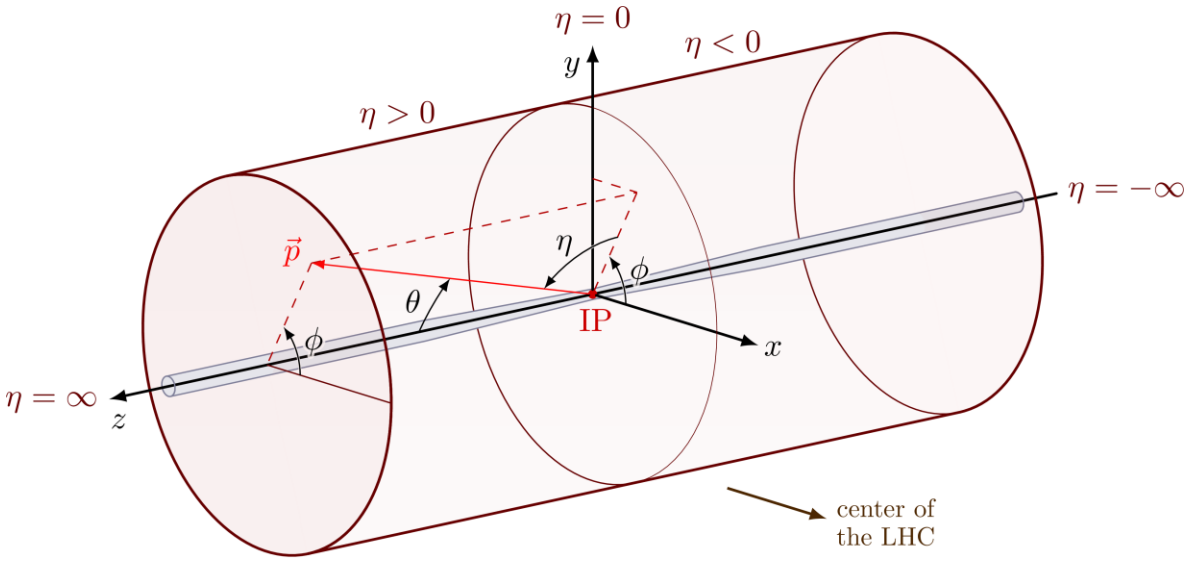


Figure 2.3. CMS coordinate system [13].

2.4. Object Reconstruction in CMS

To analyze the recorded data from a particle detector like CMS, the output signals need to be translated into meaningful physical objects. To achieve this, specific algorithms have been developed to identify and reconstruct particles, leading to the creation of lists containing electrons, muons, photons, and jets. Each candidate object undergoes a series of criteria to determine its classification within one of these groups. This chapter will provide an overview of the criteria and algorithms employed to obtain the appropriate objects for subsequent physics analysis.

2.4.1. Particle-Flow Algorithm

The Particle Flow (PF) algorithm [14] is a key feature of the CMS detector to accurately reconstruct and identify all the particles produced in proton-proton collisions with high efficiency and precision. This is achieved by effectively merging and utilizing the information provided by various subdetectors in the CMS detector. It makes use of the fact that different particles leave various signatures in the detector.

The particle flow algorithm is a multistep process that involves the combination of information from all subdetectors of the CMS detector (see Fig. 2.4). It begins by reconstructing charged particles using information from the tracker subdetector, which measures the trajectory and momentum of charged particles as they move through the

magnetic field generated by the CMS solenoid. The next step is to identify photons and electrons, which are reconstructed using information from the ECAL that measures the energy of particles through their electromagnetic interactions. Electrons and photons deposit most of their energy in the ECAL, allowing for their precise reconstruction. Hadronic particles, such as mesons and pions, are reconstructed using information from the HCAL which measures the energy of particles through their hadronic interactions.

The particle flow algorithm combines information from the ECAL and HCAL to reconstruct hadronic jets. These jets are collimated streams of particles produced through the hadronization of quarks and gluons. Finally, the remaining energy that is not accounted for by the identified particles is attributed to the presence of undetected neutral particles, such as neutrinos. This missing energy is measured using the concept of missing transverse energy, which is the negative vector sum of the transverse momentum of all identified particles. The following section provides a more detailed explanation of jet and missing transverse energy.

Once all the particles have been reconstructed and identified, the PF algorithm combines the information from all subdetectors to obtain the best possible measurement of each particle's energy and momentum. This information is then used to study the properties of the particles produced in proton-proton collisions, such as their masses, lifetimes, and interactions. The use of the PF algorithm has significantly improved the reconstruction of objects in the CMS experiment, enabling a wide range of physics analyses to be performed with high sensitivity and precision. Its accuracy and reliability are crucial for the success of the CMS physics program, enabling physicists to study the properties of particles, search for new physics phenomena BSM, and tested the predictions of theoretical models with high precision.

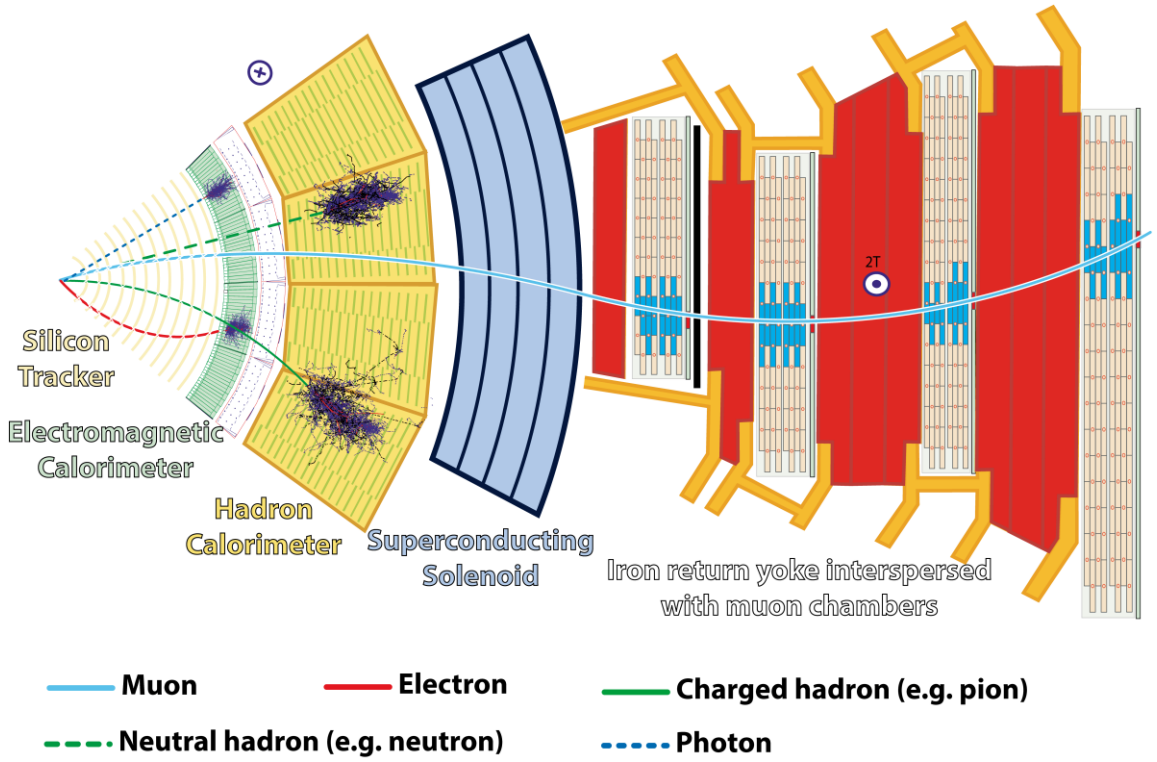


Figure 2.4 Sketch of the specific particle interactions within a transverse slice of the CMS detector, spanning from the beam interaction region to the muon detector [15].

2.4.2. Jet and MET

A jet is a concentrated cone that contains the fragmented and hadronized products of partons (quarks or gluons) following a collision. The confinement phenomenon dictates that quarks and gluons cannot exist in isolation; instead, they undergo hadronization, resulting in a cascade of colour-neutral hadrons. To reconstruct the initial parton, all the hadrons originating from it must be combined. Jet clustering algorithms are used to merge particle flow candidates in a well-defined manner. Two important criteria for a jet algorithm are infrared safety and collinear safety. Infrared safety ensures that a jet remains unchanged when soft radiation is included or excluded, while collinear safety addresses the preservation of a jet's properties in the presence of collinear splitting of particles.

Jets in particle collisions can be very messy, with hundreds of particles produced close to each other due to hadronization and the subsequent decays of unstable hadrons. Therefore, it is usually not practical to analyze the individual particles within a jet. Instead, jets are typically analyzed as a single entity, with various algorithms used to cluster the particles into the jet. In the CMS, there are different types of jet clustering algorithms available, which use different inputs from the detector to identify the particles that belong to each jet

[16]. For example, the "CaloJets" algorithm clusters particles based only on their energy deposits in the calorimeters, while "GenJets" algorithm uses simulated particles from a simulation e.g., Monte Carlo. However, the most used jet algorithm in the CMS is the "PFJets" algorithm, which is based on the particle flow technique. This algorithm takes advantage of the high granularity and excellent resolution of the CMS detector to reconstruct individual particles, including charged particles, photons, and neutral hadrons as described in the previous section. These reconstructed particles, known as "particle flow candidates", are then clustered together to form jets using an iterative jet clustering algorithm.

The "anti-kt" algorithm [17] is one of the most popular iterative jet clustering algorithms used by CMS to cluster particle flow candidates into jets. In the "anti-kt" algorithm, abbreviated as “AK”, the distance between two particles i and j is defined as:

$$d_{ij} = \frac{\min\left(p_{T,i}^{-2}, p_{T,j}^{-2}\right) \Delta R_{ij}^2}{R^2} \tag{2.5}$$

Here, $p_{T,i}$ and $p_{T,j}$ represent the transverse momenta of particles i and j, ΔR_{ij} is the distance between the particles in the $\eta - \phi$ plane, and R is a distance parameter that controls the size of the jets. The negative power of -2 used in the anti-kt algorithm ensures that particles with higher momenta are clustered first. As a result, the resulting jets tend to have a circular shape centered around the highest-momentum particle, as shown in Fig. 2.5. The algorithm proceeds by iteratively combining the closest pair of particles based on the distance measure described above, until a stopping criterion is met. When $d_{ij} < p_{T,i}^{-2}$, particles i and j are combined, and the distances are recalculated. This process continues until a point is reached where $d_{ij} > p_{T,i}^{-2}$. At this point, object i is identified as a jet and removed from the particle list. The procedure is repeated until the list of objects is empty, resulting in the formation of reconstructed jets.

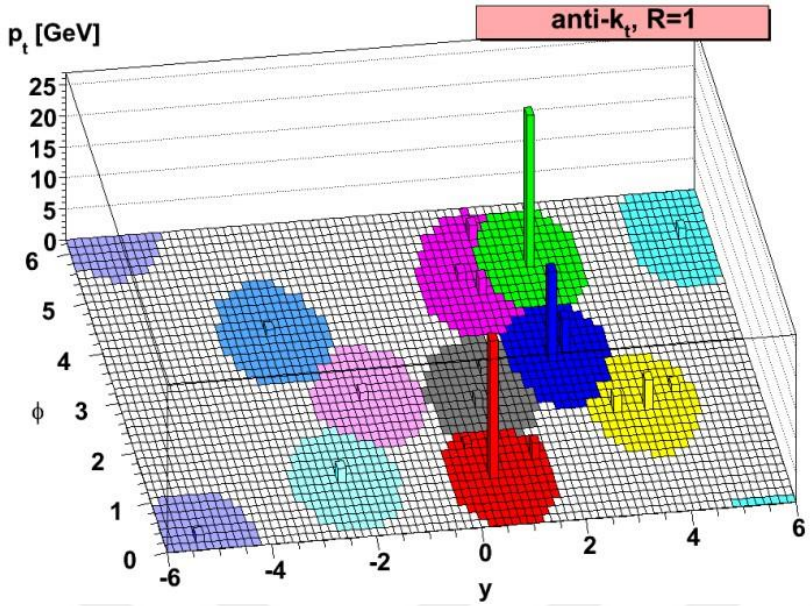


Figure 2.5. Visualization of a parton-level event clustered using the anti-kt jet algorithm, highlighting the distinctive "active" catchment areas of the resulting hard jets [17].

The CMS detector is capable of detecting the majority of stable or long-lived particles generated during pp collisions. However, neutrinos and hypothetical neutral weakly interacting particles, being exceptions, do not yield a signal in the detector. Nonetheless, their presence can be inferred by observing a momentum discrepancy in the plane perpendicular to the beam direction, known as missing transverse momentum, represented by $\vec{E}_T^{miss}$. The magnitude of this discrepancy is quantified as missing transverse energy (MET) and denoted as E_T^{miss}.

Generally, the missing transverse energy (MET) can be defined as the opposite of the vectorial sum of the transverse momenta of all the detected particles in the final state [18],

$$\vec{E}_T^{miss} = -\sum_{all\ particles} \vec{p}_T \tag{2.6}$$

CMS has developed three distinct algorithms to reconstruct the MET from the detected particles in the event. These algorithms include PF MET [19], which utilizes the particle flow technique; Calo MET [20], which relies on the energy deposits within the calorimeters; TC MET [21], which enhances the Calo MET by incorporating tracks reconstructed within the inner tracker while accounting for the expected energy depositions of those tracks in the calorimeter. The choice of MET algorithm depends on the specific analysis goals and the

available detector information. However, the PF MET algorithm is the most common algorithm in CMS due to its high performance and robustness.

In MET analysis, measurements always come with some level of uncertainty. This can be due to a variety of factors, such as object mis-reconstruction, detector noise, or finite detector resolution. These uncertainties can contribute to the measured MET in an event, making it difficult to determine whether the observed MET is the result of a real physical process or just a statistical fluctuation (fake MET). To address this issue, a missing transverse energy significance variable has been developed and denoted as "MET Significance" or METSig [18]. The significance evaluates, on an event-by-event basis, the probability that the observed MET is inconsistent with a null hypothesis MET = 0, considering the full event composition and resolution functions for each object in the event.

MET significance is defined as the ratio of the observed MET to its uncertainty, expressed in units of standard deviations. This ratio can be thought of as a measure of how many standard deviations the observed MET deviates from the expected MET due to statistical fluctuations and detector limitations. The formula for calculating MET significance is as follows,

$$METSig = \frac{MET}{\sqrt{\sum|\vec{E}_T|}} = \frac{MET}{\sqrt{\sum|\vec{p}_T|}} \tag{2.7}$$

Where MET is the observed missing transverse energy in the event, and $\sum|\vec{E}_T|$ is the scalar sum of the transverse energies of all the particles detected in the event. The square root of $\sum|\vec{E}_T|$ represents the expected statistical uncertainty in the measurement of MET due to the finite precision of the detector. In general, a high value of METSig suggests that the observed MET in the event cannot be fully explained by resolution smearing alone, indicating the presence of unseen objects like neutrinos or other exotic weakly interacting particles [18].

2.5. The CMS DQM and DC

CMS produces a vast amount of data, which is used to study the properties of particles and their interactions. The accuracy and integrity of this data are of utmost importance. Any errors or inconsistencies in the data can lead to incorrect conclusions or even false

discoveries. Hence, it is vital to certify the CMS data as suitable for physics analysis to guarantee the accuracy and quality of all physics results released by the CMS Collaboration. For this reason, an integrated monitoring system called Data Quality Monitoring [22] was established, comprising processes, methods, and software infrastructure. The DQM system enforces strict quality criteria to ensure that only high-quality data is used for physics analysis.

There are several types of reconstruction workflows used in CMS, each with different levels of sophistication and accuracy. Three of the most used types are Prompt-reco, Re-reco, and Ultra-legacy. The DQM system monitors and validates all workflows, including the particle flow algorithm used in the three types. Prompt-reco refers to the reconstruction of data in real-time as they are recorded by the CMS during LHC operations. The reconstructed data are quickly made available for physics analysis, typically within a few days of the data being taken. Re-reco stands for "re-reconstruction" and refers to a more refined reconstruction of the data performed after the initial prompt-reco. It is typically used to improve the accuracy of the reconstructed objects and to take advantage of improved calibration and alignment information that becomes available as more data is recorded. Ultra-legacy reconstruction refers to the most sophisticated and accurate reconstruction of CMS data, using the latest calibration and alignment information and the most advanced reconstruction algorithms. Ultra-legacy reconstruction is typically performed on the full dataset, after several years of LHC operation, and provides the most precise measurement of physical observables. The particle flow algorithm, which is used in all three workflows, is also monitored, and validated by the DQM system. All workflows, including the particle flow algorithm used in Prompt-reco, Re-reco, and ultra-legacy, are monitored and validated by the DQM system.

In CMS, the particle collision data is organized into runs, which correspond to specific configurations of both the CMS detector and the LHC accelerator. The duration of a run can vary, lasting from a few minutes up to approximately 20 hours. To effectively handle the data within a run, it is further divided into Lumisections (LSs). A LS represents a time interval during which the CMS detector continuously gathers data, assuming that the instantaneous luminosity remains constant. Each LS encompasses around 23 seconds of data acquisition and contains hundreds or thousands of collisions, referred to as events, which consist of recorded particle hits. Typically, the analysis is not conducted on

individual events but rather on a collection of events grouped together based on the accurate detector response during specific time periods. Each event can be uniquely identified by specifying the event number, the LS number, and the run number [23].

The DQM group collaborates with each subsystem of CMS to gather data concerning the respective subsystem and present it in a meaningful manner. Subsystems are typically composed of the various detectors used, but they can also include groups that are not directly represented by the physical detector, such as the trigger systems and the DQM infrastructure. The output of the DQM takes the form of numerous plots, referred to as monitor elements, which are typically histograms in ROOT [24] format. These histograms are derived from event data pertaining to specific sub-partitions of the detector, collected over a given time window. The monitor elements are typically organized according to the different subsystems in the CMS experiment, allowing experts to focus on specific aspects of the detector's performance.

The DQM is performed online and offline. During online monitoring, real-time DQM is carried out during data acquisition to identify any issues and provide immediate feedback on the detector status. This is done by evaluating the quality of data during the data-taking process for LSs. On the other hand, offline monitoring involves promptly reconstructing, monitoring, and certifying the data, typically with a delay of approximately 48 hours from when the collisions were recorded. The primary goal of offline monitoring is to ensure that the data is of sufficient quality for use in physics analysis, and it is evaluated on a per-run basis.

For online and offline data, shifters are in charge of monitoring the results and delivering a manual certification. They assign a Boolean flag (green for good and red for bad) for each subsystem. The manual certification process is based on these flags. Good runs refer to those where all subsystems operated correctly, and the data can be used for physics analysis without further scrutiny. On the other hand, bad runs include those where only a subset of the subsystems malfunctioned. In such cases, the relevant subsystems are checked to determine whether they are usable or not for specific analyses. Runs that are not completely bad will be identified with a dedicated keyword in the comment stored in the Run Registry [25], which is a database that keeps track of the status of each run and the certification decisions made by the shifters, as illustrated in Fig. 2.6. The database contains information

about each run, such as the run number, the LS number, the data-taking conditions, and the status of the various subsystems during that run.

The DQM system in CMS operates at multiple levels, and each level corresponds to a user group with a distinct purpose [26]. At the Detector Performance Group (DPG), the status and performance of each subsystem are monitored up to the local reconstruction stage, and this monitoring is done both online and offline. The Physics Object Group (POG) is responsible for monitoring the quality of reconstructed physics objects such as muons, jets, and electrons, but it only operates during the offline stage. Finally, at the Physics Analysis Group (PAG), higher-level quantities such as dilepton mass peaks and correlations between different physics objects are monitored, along with kinematic distributions that have more analysis-oriented cuts. Like the POG, the PAG operates only during the offline stage of data analysis. The focus of this thesis is on the JetMET POG Group (JMEPOG), which is responsible for monitoring the quality of the reconstructed jets and MET in CMS.

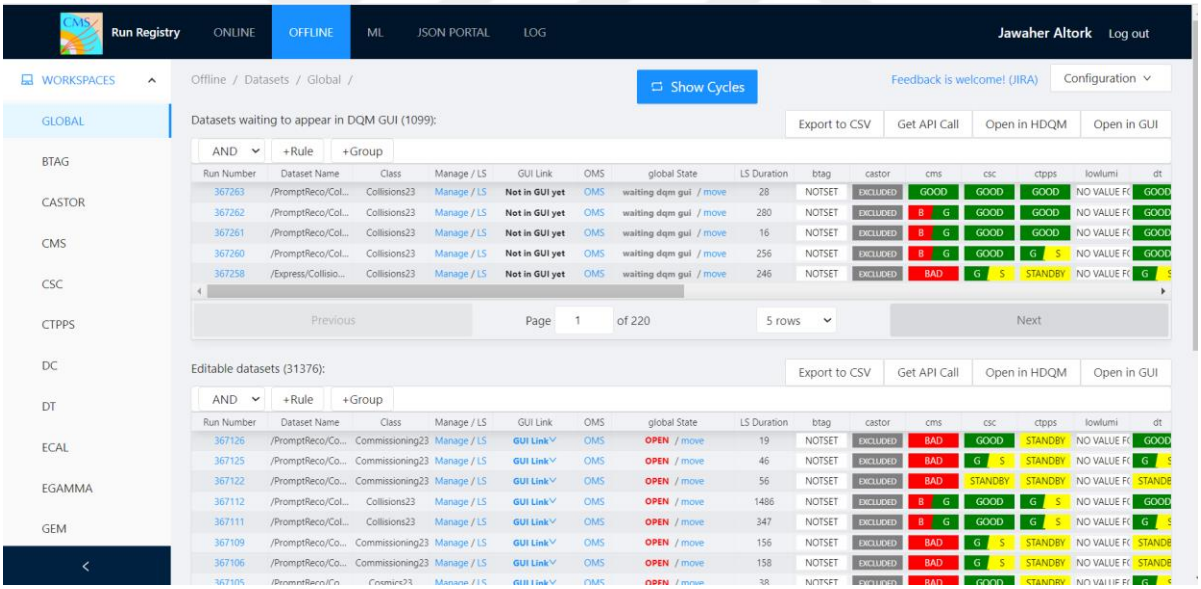


Figure 2.6. A snapshot of the CMS DQM Run Registry [25].

The DQM system plays a crucial role in the CMS readout and reconstruction chain, encompassing various components supported by highlighted software components shown in grey in Fig. 2.7. The DQM code is executed within the CMS Software (CMSSW) environment, which includes the High-Level Trigger (HLT), online DQM machines, data reconstruction jobs, as well as simulation and reconstruction jobs for release validation and Monte-Carlo production. The output histogram monitor elements are aggregated in the DQM Graphical User Interfaces (GUIs) [27], which store the data and provide a web-based

interface for browsing and visualizing the DQM output. The DQMGUIs are accessed by DQM shifters, certification, and validation teams, as well as subdetector experts. The certification team makes determinations based on the DQM output and records them in the Run Registry. Then, a golden JSON file is generated, which lists the data suitable for physics analysis and includes all runs with certified data quality.

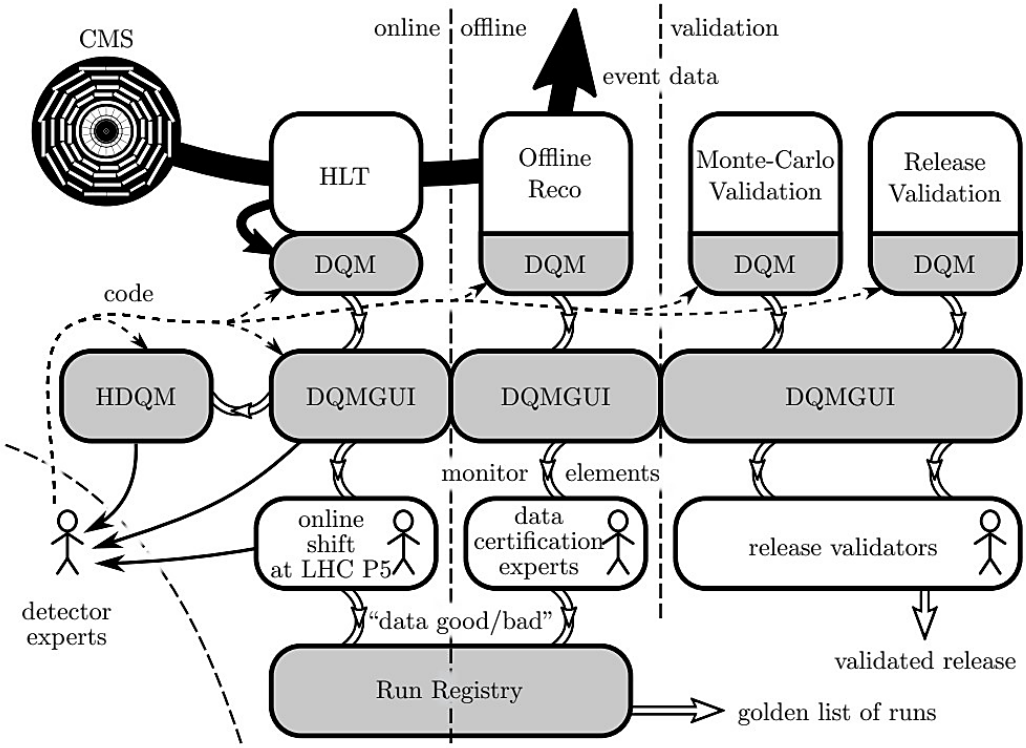


Figure 2.7. Schematic overview of the DQM systems [23].

In the DQM GUI, there are different types of jets that are used in different types of analyses depending on the physics process being studied and the requirements of the analysis. It includes:

1. Cleanedak4CaloJets: These jets are reconstructed using a clustering method on the energy deposits in the calorimeter subsystem of the detector. The term "cleaned" in the name refers mainly to the usage of detector control system flags of sub-detectors in the JetMET DQM framework. These flags help remove any spurious energy deposits, such as those caused by electronic noise or particles not associated with the primary collision. The purpose of this cleaning process is to ensure that the collection of jets consists of pure collision data without any detector inefficiencies.

2. Cleanedak4PFJets: These are jets that are reconstructed using a clustering algorithm on the charged and neutral particles that are reconstructed using the PF technique in the detector. The "cleaned" in the name refers to the fact that these jets have undergone cleaning algorithms to remove particles not associated with the primary collision.

3. Cleanedak4PFJetsCHS: These are jets that are reconstructed using the same clustering algorithm as Cleanedak4PFJets, but with the additional application of the Charged Hadron Subtraction (CHS) technique to reduce the effects of pileup. This technique subtracts the energy and momentum associated with charged particles that are not originating from the primary vertex, which helps to improve the energy resolution of the jets.

4. Uncleanedak4PFJets: These are jets that are reconstructed using the same clustering algorithm as Cleanedak4PFJets, but without any additional cleaning or subtraction techniques applied. These jets may contain spurious energy deposits or particles not associated with the primary collision and are typically used for comparison or background estimation purposes.

Where the AK4 refers to anti-kT algorithm with a distance parameter of 0.4, which is used for clustering jets in both the calorimeter and tracker [28].

One aspect of data certification is the certification of quantities related to hadronic jets and MET. These are important quantities in many physics analyses carried out by the CMS collaboration and are the quantities of interest for this thesis. The current method for certifying these quantities is largely dependent on manually monitoring reference histograms that summarize the status and performance of the detector. These histograms are created using simulated data or real data taken during calibration runs where known processes are studied. They represent the expected behaviour of the detector in the absence of any issues and the data is compared to these histograms to identify any discrepancies or anomalies as seen in Fig. 2.8 below. Any deviations from the expected behaviour are flagged and investigated further. This process is currently carried out by trained shifters who monitor the histograms and compare the data to the reference histograms. But the increasing complexity of the detector, high data volumes, and need to manage different LHC running scenarios require greater automation. The number of histograms to monitor

is already challenging for the shifters, and detecting new failure modes requires more data and increases detection delay. The cost of human resources, including 24/7 DQM shifters and experts responsible for updating good data references, is also substantial. Adopting automated anomaly detection (AD) in the current LHC Run is the practical way to prepare for these challenges and grow expertise in developing and operating various machine learning models and tools. Therefore, this thesis represents a crucial step towards introducing advanced machine learning models for AD in the sophisticated monitoring infrastructure of the CMS experiment.

In the context of JetMET data certification, CMS has recently introduced an autoencoder based anomaly detection tool capable of detecting anomalies in DQM monitor elements with a per-Lumisection granularity [29]. This has led to a boost in the efficiency of the online DC procedure. The study presented in this context concentrates on using machine learning techniques to certify the runs of the DQM system, with a focus on hadronic jet and MET elements. Automating the offline DC procedure is expected to enhance its efficiency further. The two approaches will work together in a complementary manner; the offline approach will help to identify Runs with anomalous data, which can then be further analyzed using the online approach to identify the specific subset of LSs that contain the anomalies.

This dissertation discusses the current progress in using machine learning for automating anomaly detection and aiding humans in the certification process for JetMET. Specifically, we explore various supervised and unsupervised techniques, including neural networks and autoencoders, which are detailed in the following chapter.

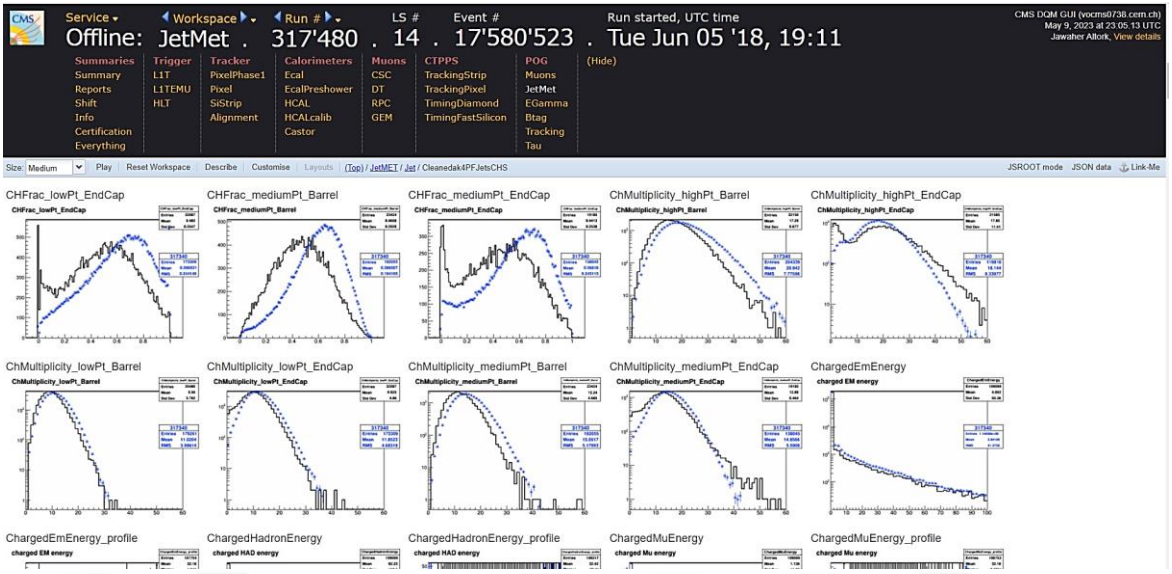


Figure 2.8. Snapshot of the CMS Graphical User Interface used for the visualization of the data quality information. It shows how offline DQM works where run 317480 (black) is flagged bad according to the reference good run 317340 (blue) [27].

3. Machine Learning Algorithms

Machine learning (ML) falls under the domain of artificial intelligence (AI) and is dedicated to the development of computer algorithms and models capable of enhancing their performance on specific tasks or problems without explicit programming. ML algorithms and models utilize statistical analysis, pattern recognition, and inference techniques to autonomously detect patterns and relationships within data, enabling them to generate predictions or recommendations. Instead of relying on manual instructions, ML leverages data-driven processes to facilitate automated learning and decision-making. In this thesis work, various Python-based libraries such as Scikit-Learn [30], TensorFlow [31], and Keras [32] will be used, which are widely used tools for implementing machine learning algorithms and models. These libraries provide a wide range of pre-built algorithms and models, as well as tools for data preprocessing, feature extraction, model evaluation, and visualization.

In ML, data can be categorized into two main types: labelled data and unlabeled data. Labelled data refers to data that includes both the input features and the corresponding target values, or labels. This type of data is typically used for supervised learning algorithms, where the goal is to learn a mapping between the input features and the output labels. One example of a supervised learning task that will be utilized in this thesis is

classification, where data is labeled, and a model is trained to predict the class of new, unseen data based on the given labels. Unlabeled data, on the other hand, refers to data that does not include the target values. This type of data is typically used for unsupervised learning algorithms, where the goal is to discover patterns or structures in the data without any pre-specified labels. One example of an unsupervised learning task that will be utilized in this thesis is anomaly detection, where the model is trained on a dataset without labeled data and tasked with identifying unusual or anomalous observations that deviate from the typical patterns in the data. In the following sections, both supervised and unsupervised machine learning approaches will be discussed and explained in detail.

3.1. Supervised Learning

Supervised learning is the approach that will be discussed first, with a focus on binary classification. The goal is to predict one of two possible categories or classes for a given input data point. For the offline JetMET DQM, binary classification is used to certify runs as either good or bad. To accomplish this, three distinct classifiers from the scikit-learn [30] library are used; K-nearest neighbors (KNN), Gaussian Naive Bayes (GNB), and Support Vector Machine (SVM).

Before we train the model on the data it is necessary to normalize. This step is crucial because certain machine learning algorithms are particularly sensitive to the varying scales of input features. For instance, features with larger numerical values have the potential to overpower the learning process, thereby impeding the algorithm's ability to effectively learn from other features. In the context of the scikit-learn classifiers, normalization can be performed using the Standard Scaler module, which scales each feature to have zero mean and unit variance,

$$q_{norm} = \frac{q - \mu}{\sigma} \tag{3.1}$$

Where q is the data point, μ is the mean and σ is a standard deviation from the mean. This ensures that each feature has a similar scale and prevents one feature from dominating the others during the learning process.

3.1.1. K- Nearest Neighbours

The Nearest Neighbour Classifier [33] is recognized as one of the most straightforward classifiers in machine learning. Its principle is simple: examples are classified based on the class of their nearest neighbours. To extend the classification process beyond a single neighbour, the technique is commonly known as K-Nearest Neighbour Classification, where K denotes the number of nearest neighbours considered to determine the class. By considering multiple neighbours, the KNN algorithm expands upon the basic idea of the Nearest Neighbour Classifier and offers increased flexibility in classification decisions.

There is no predefined statistical method for determining the optimal value of K. However, cross-validation method can be used to estimate the best value of K. This involves training the classifier multiple times with different values of K and selecting the value that minimizes the number of errors while maintaining accurate prediction capability. Additionally, in binary classification tasks like the one in this study, it is preferable to choose an odd value for K to avoid ties in samples at the decision boundaries.

The basic idea is, as shown in Fig. 3.1, a 3-nearest neighbour classifier where the classification is based on the three closest neighbours to a given point in a two-dimensional feature space. In this example, the training data consists of runs that are labelled as either good or bad, based on their respective mean values of the Charged Hadron Energy Fraction (CHF) for Endcap and Barrel detectors. The decision for run (q1) is straightforward, as all three of its nearest neighbours are bad runs. Therefore, run (q1) is classified as a bad run. For run (q2), however, the situation is more complicated. It has one neighbour that is good and one that is bad. To resolve this, simple majority voting is used. The classifier finds the three closest training points to the testing point, based on some distance metric such as Euclidean, Manhattan, or others. In this work, Euclidean distance is used,

$$d = \sqrt{\sum_{k=1}^{n} (a_k - b_k)^2} \tag{3.2}$$

where n is the number of dimensions and a_k and b_k are the k-th components of the data objects (testing samples and training samples, respectively).

The Euclidean distance between a testing point and each of the training points is calculated, and the three closest training points are selected based on the shortest distances. The class

label is then predicted through majority voting, where the class label that appears most frequently among the three closest training points is chosen as the runs, so it is classified as a good run.

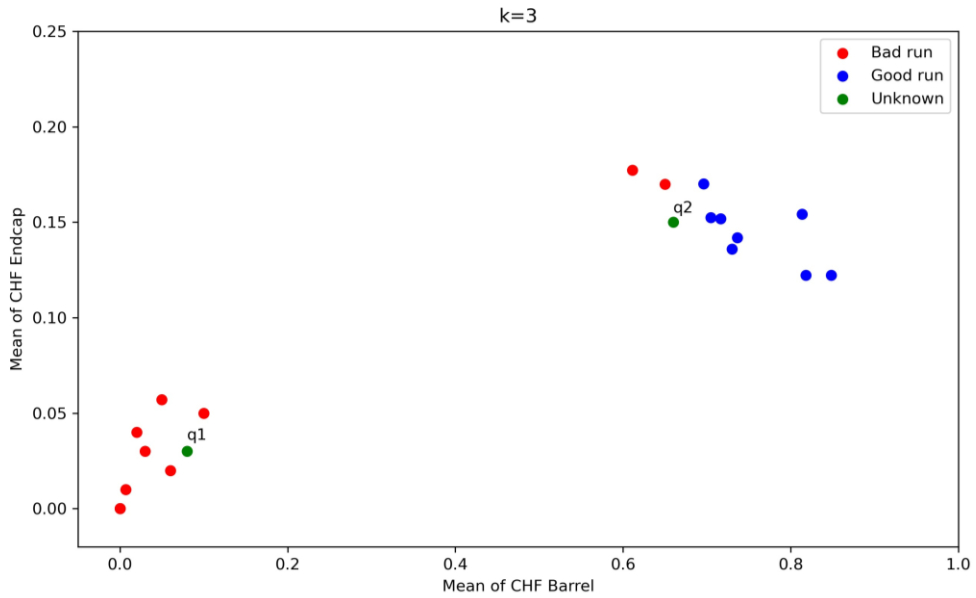


Figure 3.1. A 3-Nearest Neighbors Classifier trained on a two-class problem in a two-dimensional feature space.

KNN classification consists of two stages: determining the nearest neighbours and determining the class using those neighbours. In the following example, KNN is applied to a larger dataset of runs. Fig. 3.2 depicts two classes of good and bad runs, represented by blue and red colours, respectively. After applying the KNN classifier (see Fig. 3.3), the region is classified into different colours. Runs in red areas are most likely to be classified as bad, while runs in blue areas are classified as good. Runs in white areas have a 50% probability for both classes.

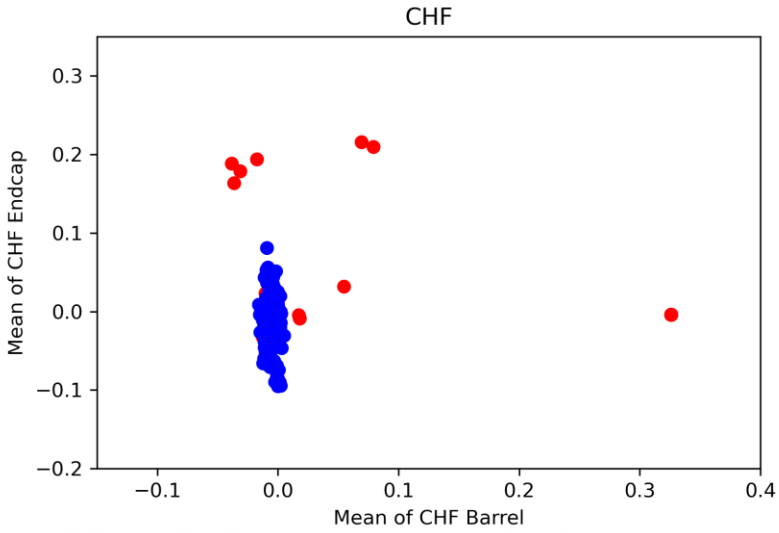


Figure 3.2. Two classes of runs are represented by the colors blue and red for the mean values of CHF, where blue represents good runs and red represents bad runs.

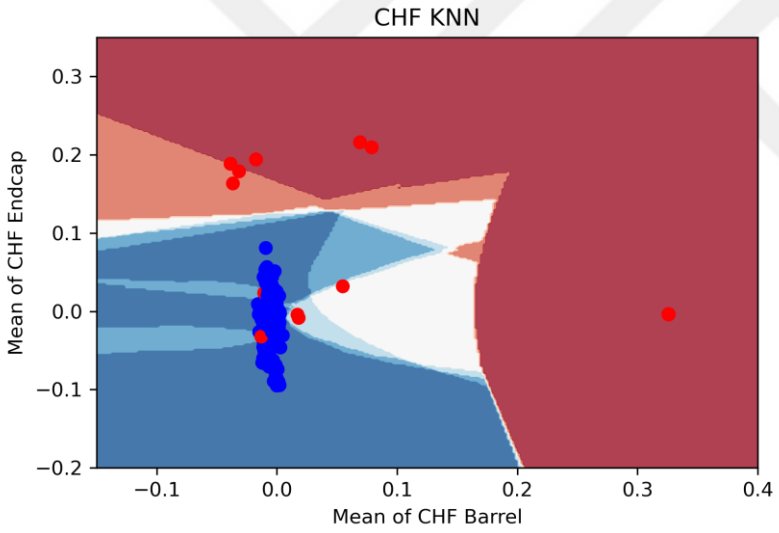


Figure 3.3. KNN classifier trained on the mean values of CHF for a set of good and bad runs.

3.1.2. Gaussian Naïve Base

Gaussian Naive Bayes (Gaussian-NB) is a variant of Naive Bayes [34] that follows Gaussian normal distribution and supports continuous data. It is a probabilistic classifier that uses Bayes' theorem to estimate the class probabilities of a given data point. Bayes' theorem can be expressed mathematically as,

$$p(t|x) = \frac{p(x|t)p(t)}{p(x)} \tag{3.3}$$

Where $p(t)$ is the probability of t occurring, $p(x)$ is the probability of x occurring, and $p(x|t)$ is the probability of x given t.

The classifier assumes that the features of a data point are independent of each other and normally distributed, hence the term "Gaussian". One of the advantages of the GNB classifier is that it requires a small amount of training data to estimate the parameters of the model. It is also computationally efficient and can handle missing data. However, it may not perform well on datasets with highly correlated features or when the assumption of normality is violated.

Naive Bayes Classifiers are built upon the principles of Bayes' Theorem, which enables the calculation of conditional probabilities between two events based on the individual probabilities of each event. One key assumption made by these classifiers is the strong independence assumption among the features. They assume that the value of one feature is independent of the values of other features. Naive Bayes classifiers can efficiently estimate the necessary parameters for classification using small training datasets. In the case of continuous data, it is often assumed that the continuous values associated with each class follow a normal (or Gaussian) distribution. The likelihood of the features is assumed to be Gaussian, and the conditional probability is given by:

$$p(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{\left(x_i - \mu_y\right)^2}{2\sigma_y^2}\right) \tag{3.4}$$

Where x_i is the i-th feature of the input vector, y is the class label, μ_y is the mean of the i-th feature for the training samples of class y, σ_y^2 is the variance of the i-th feature for the training samples of class y, and π is the mathematical constant.

Naive Bayes classifiers, despite their simplified assumptions, have demonstrated effective performance in numerous classification problems. They are known for their speed and simplicity compared to more complex methods. However, their main limitation is the assumption of feature independence. This assumption may pose challenges in certain cases, but in binary classification scenarios where the predictors are independent, such as the dataset we employed, it does not present a significant issue.

The classifier functions by computing the probability of a data point belonging to each class, given its feature values. Subsequently, the class exhibiting the highest probability is designated as the predicted class label for that specific data point. The example below (see Fig. 3.4) shows how the Gaussian-NB classifier can be applied to a dataset of runs. Using the same dataset as for the previous KNN approach, the classifier classifies the region into different colors. The red areas indicate runs that are most likely to be classified as bad, while the blue areas represent runs classified as good. The white areas have a 50% probability for both classes.

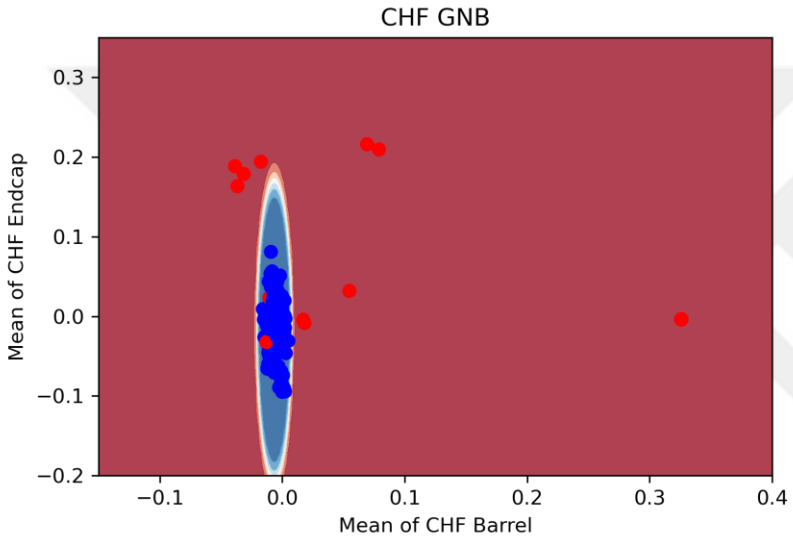


Figure 3.4. Gaussian-NB classifier trained on mean values of CHF for a set of good and bad runs.

3.1.3. Support Vector Machine

Support Vector Machine (SVM) [35] is a powerful algorithm that is based on simple concepts. It works by representing different classes in a hyperplane within a multidimensional space. SVM iteratively generates the hyperplane in a way that minimizes the error. The primary objective of SVM is to separate the dataset into different classes and identify a Maximum Marginal Hyperplane (MMH). The followings are important concepts in SVM:

1. Support Vectors: These data points represent the closest points to the decision boundary or hyperplane. These points are critical for defining the hyperplane and the margin.

2. Hyperplane: it is the decision boundary or separating line that divides the data points into different classes. The SVM algorithm aims to find the hyperplane that maximizes the margin between the classes.
3. Margin: it is the distance between the hyperplane and the closest data points of each class. A larger margin is desirable as it indicates better generalization performance of the SVM model. The support vectors lie on the margin, and the SVM algorithm tries to maximize this margin.

The primary objective of SVM is to effectively divide a given dataset. The margin, which represents the distance between the nearest point, plays a crucial role in this process. The goal is to identify a hyperplane that maximizes the margin between the support vectors in the dataset. SVM follows a series of steps to search for the hyperplane with the maximum margin. Initially, multiple hyperplanes were created to separate the classes in the most optimal way possible. In Fig. 3.5, three hyperplanes (black, blue, and orange) are depicted on the left-hand side. While the blue and orange hyperplanes exhibit higher classification errors, the black hyperplane accurately separates the two classes. Subsequently, the appropriate hyperplane is chosen based on its maximum separation from the nearest data points, as illustrated in the figure on the right-hand side.

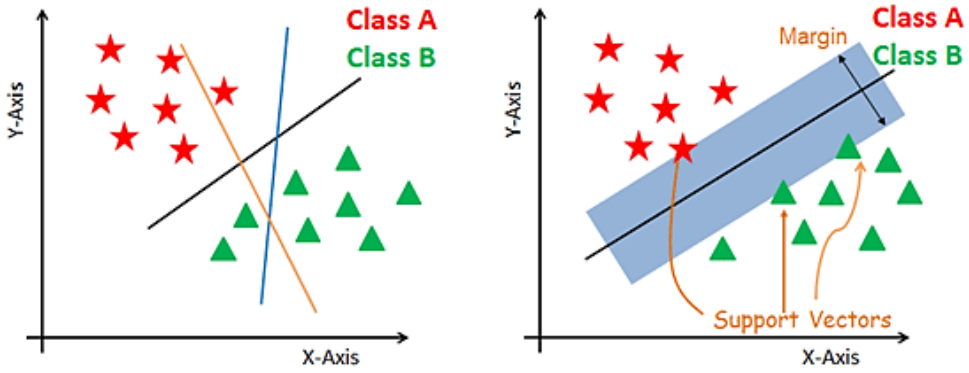


Figure 3.5. An example to show how the SVM classifier works to find the best possible hyperplane that separates the red and green data points [36].

The SVM algorithm is implemented in practice using a kernel. This is achieved through a technique known as the kernel trick, which takes a low-dimensional input space and maps it into a higher-dimensional space. By doing so, SVM can handle non-separable problems by adding more dimensions to make them separable. The kernel trick plays a crucial role in constructing a more precise classifier. In this study, the Radial Basis Function (RBF)

kernel [37] is employed as the kernel function. The RBF kernel computes the similarity between two data points in a high-dimensional space using a Gaussian function. The RBF kernel is defined as,

$$K(x_i, x_j) = e^{\left(-\gamma (x_i - x_j)^2\right)} \tag{3.5}$$

Where x_i and x_j are the input feature vectors, and γ is a hyperparameter that determines the smoothness of the decision boundary.

So, the SVM classifier operates by mapping the data into a higher-dimensional feature space, enabling the categorization of data points even when they are not linearly separable. The algorithm constructs a hyperplane or a set of hyperplanes in this high-dimensional space, facilitating the classification of new data points.

The example shown in Fig. 3.6 demonstrates how the SVM classifier can be applied to a dataset of runs, using the same dataset as the previous classifiers. As shown in the resulting classification, the SVM classifier can accurately classify the region into different colours. The red areas indicate runs that are most likely to be classified as bad, while the blue areas indicate runs that are classified as good.

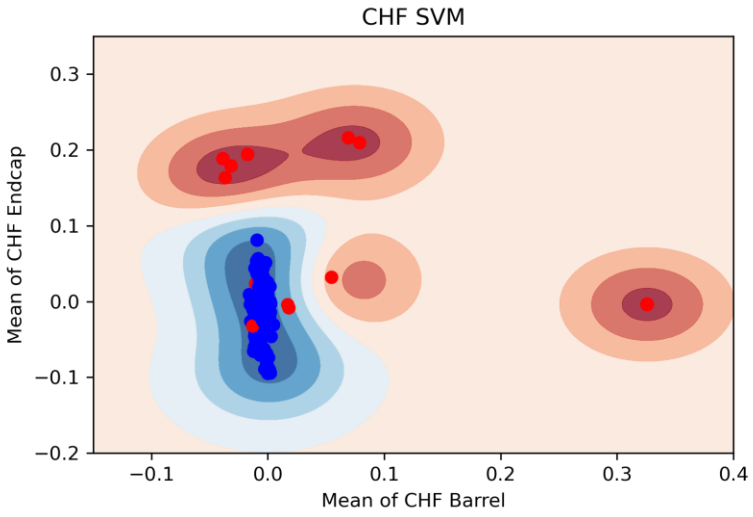


Figure 3.6. SVM classifier trained on mean values of CHF for a set of good and bad runs.

3.2. Neural Networks

Neural Networks [38], also referred to as Artificial Neural Networks (ANNs), are a subset of machine learning techniques. They derive their name and structure from the human brain, aiming to mimic the signalling process between biological neurons. Neural networks can employ either supervised or unsupervised learning algorithms. Typically, a neural network consists of interconnected units called neurons, arranged in a series of layers that establish connections between adjacent layers. These networks are designed to receive various types of information, which the network aims to learn, recognize, or process. On the opposite side of the network, there are output units that indicate the network's response to the learned information. Between the input and output units, additional layers of hidden units are added, forming the majority of the artificial brain (see Fig. 3.7). Various neural network structures have been developed for tasks such as image reconstruction and binary classification. In this study, the employed neural network structures include fully connected neural networks and autoencoders.

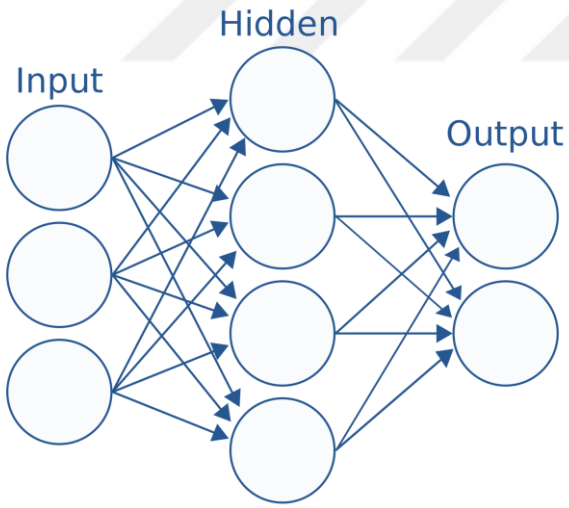


Figure 3.7. Simple architecture of neural networks with an input layer, a hidden layer, and an output layer [39].

Fully Connected Neural Networks (FCNNs) are widely used in neural network architectures, where each hidden unit and output unit is connected to every unit in the adjacent layers. These connections are characterized by weights, which determine the influence of one unit on another. In FCNNs, each neuron (or perceptron) represents an individual function within a network. Within fully connected layers, the neuron applies a linear transformation to the input vector using a weight matrix. Subsequently, a non-linear

activation function f is applied to the resulting product, enabling the network to incorporate non-linear relationships and capture complex patterns in the data. A non-linear transformation is then applied to the product through a non-linear activation function f.

$$y_{jk}(x) = f\left(\sum_{i=1}^{n} w_{jk}x_i + w_{j0}\right) \tag{3.6}$$

Here, the dot product is computed between the weight's matrix w and the input vector x. The bias term (w_0) can be incorporated within the non-linear function. The activation function f wraps the dot product between the input of the layer and the corresponding weight matrix. Where the columns in the weight's matrix will all have different numbers and will be optimized as the model is trained.

The activation function plays a crucial role in neural networks by transforming the weighted sum of inputs with bias into an output within a specified range. It introduces non-linearity, which enables the network to learn complex patterns and relationships in the data. Without an activation function, the network would behave similarly to a linear regression model with limited learning capabilities. In this study, the following activation functions are used:

1. Sigmoid function [40]

The sigmoid function, often represented as $\sigma(x)$, is a special form of the logistic function. Sigmoidal units possess the characteristic of mapping the entire range of number lines onto a small interval, typically between 0 and 1. Consequently, one of the key applications of a sigmoid function is to transform a real-valued input into a value that can be interpreted as a probability. Mathematically, it is expressed as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.7}$$

And its derivative is:

$$\frac{d}{dx}\sigma(x) = \sigma(x)\big(1 - \sigma(x)\big) \tag{3.8}$$

2. Tanh function [40]

The hyperbolic tangent function, denoted as tanh, compresses a real-valued number into a range spanning from -1 to 1. Similar to the sigmoid function, it is a non-linear activation function. However, unlike the sigmoid function, the output of tanh is centred around zero. This zero-centred property makes the tanh non-linearity more commonly favoured in practice over the sigmoid nonlinearity. It's expressed as:

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.9}$$

And its derivative is:

$$\frac{d}{dx} tanh(x) = 1 - tanh^{2}(x) \tag{3.10}$$

3. SELU [41]

SELU, or scaled exponential linear unit, is an activation function that promotes self-normalization within neural networks. The use of SELU activation functions leads to neuronal activations that naturally converge to a mean of zero and a variance of one. This self-normalization property helps in stabilizing and improving the performance of the network during training. It's defined as:

$$f(x) = \lambda x\,, \qquad if\ x > 0 \tag{3.11}$$

$$f(x) = \lambda\alpha(e^{x} - 1), \qquad if\ x \leq 0 \tag{3.12}$$

Where x is the input value, $f(x)$ is the output result, and λ and α have the approximate values of 1.507009873 and 1.673263242, respectively.

As mentioned in the previous section, normalization is an important step in preparing data for training machine learning models. Adding a Batch Normalization layer [42] to a neural network is a common technique to achieve this. The layer normalizes the input data by subtracting the mean and dividing by the standard deviation, which maintains mean value close to 0 and standard deviation close to 1. This layer helps stabilize the training process and improve the performance of the model.

Deep neural networks with a large number of parameters are highly effective machine learning systems. However, overfitting poses a significant challenge for such networks. Overfitting occurs when a model learns the training data too well and starts to memorize it instead of generalizing it to new, unseen data. Accurate predictions require neural network models to exhibit robust generalization across various data types within their domain. Overfitting can arise from several factors, including inadequate training data size, where the dataset lacks sufficient samples to represent the full range of possible input data values; presence of noisy data, which refers to irrelevant or erroneous information in the training data; prolonged training on a single dataset, causing the model to overly memorize the specific samples; and high model complexity, enabling the model to inadvertently learn noise present in the training data.

One effective way to prevent overfitting is using dropout layers [43]. Dropout is a regularization technique, and its basic idea is that, during training, each neuron has a certain probability of being dropped out (i.e., set to zero) or "turned off," with each training iteration. This means that the neurons that are dropped out do not contribute to the forward or backward pass of the training process, which makes it harder for the model to memorize the training data. By randomly dropping out neurons, dropout helps to prevent co-adaptation among neurons and encourages the network to learn more robust and generalizable features. So, to implement a dropout layer, a dropout ratio which is in the range of 0 and 1 is provided, where 1 means no dropout and 0 means no output from the layer. The percent of neurons to drop out used in this study is 0.1 or 10 percent. During inference (i.e., when making predictions on new, unseen data), dropout is turned off and all neurons are used. This means that the model is able to use all of the information it has learned during training to make predictions.

After defining the model, compiling it should be next. Compilation is a step that transforms the simple sequence of layers that we previously defined into a highly efficient series of matrix transformations. Compilation can be interpreted as a precompute step that makes it possible for the computer to train the model.

To compile a neural network, some additional properties are required to be specified when training the network. The compile function in TensorFlow requires three parameters: the optimizer, the loss function, and the metric parameter.

Starting by selecting the optimizer. We will be using the Adam optimizer [44] for this work, which is a highly performant stochastic gradient descent algorithm. Adam is a popular optimization algorithm used in deep learning to update the weights of a neural network during training. It stands for Adaptive Moment Estimation and is a combination of two other optimization algorithms: Adagrad [45] and RMSprop [46]. The Adam optimizer has several hyperparameters, the most important one is the learning rate. The learning rate determines how fast the weights are updated. In other words, the optimizer determines how the weights and biases in the network are updated during training, while the learning rate controls the step size of the updates.

For this work, different loss functions are used for different datasets. Starting with binary cross-entropy loss [48] which is a commonly used loss function in ML for binary classification problems. It measures the differences between the predicted probability distribution and the actual probability distribution of the binary classification problem. In binary classification, the output is either 0 or 1, which in the case of this study, indicates whether the run is good or bad, i.e., the class to which the input belongs. Binary cross-entropy loss is calculated as,

$$\text{Loss} = -\frac{1}{n}\sum_{i=1}^{n} y_i \,.\, \log(\tilde{y}_i) + (1 - y_i) \,.\, \log(1 - \tilde{y}_i) \tag{3.13}$$

Where y is the true label (either 0 or 1) and $\tilde{y}$ is the predicted probability of the positive class. If $y = 1$, the first term contributes to the loss, and if $y = 0$, the second term contributes to the loss. This loss function encourages the model to correctly predict the class label for each input, while also providing a gradient that can be used to update the model's parameters during training. The binary cross-entropy loss function is commonly used in neural networks with a sigmoid activation function in the output layer, which ensures that the output is between 0 and 1 and can be interpreted as a probability [49].

The second loss used is the Mean Squared Error (MSE). The MSE loss function is commonly used in unsupervised autoencoder models for reconstruction tasks (see next section). It measures the average squared difference between the predicted and actual values of the output variable. Given a set of n samples, where y is the true output value and $\tilde{y}$ is the predicted output value, the mean squared error is calculated as,

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2 \tag{3.14}$$

The squared term in the loss function means that larger errors contribute more to the loss than smaller errors. During training, the model's parameters are adjusted to minimize the MSE loss function, resulting in better predictions for the output variable. However, the MSE loss function can be sensitive to outliers in the data, as large errors from outliers contribute significantly to the loss. An outlier is a data point that differs significantly from other data points in the dataset. In such cases, other loss functions such as the mean absolute error may be preferred.

So, the next loss function used in this study is the Mean Absolute Error loss (MAE). MAE calculates the absolute differences between predicted and actual values and averages these differences. The formula for MAE is:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \tilde{y}_i| \tag{3.15}$$

MAE is a more robust loss than MSE because it is less sensitive to outliers. In contrast to MSE, MAE treats all data points equally, regardless of their distance from the predicted value. A lower value of the loss indicates better performance of the model, with zero being the ideal value.

The last parameter that needed to be specified is the metric. The most common metric used to evaluate the performance of the model is the accuracy. It measures the proportion of correct predictions made by the model over the total number of predictions. It is calculated as the ratio of the number of correct predictions to the total number of predictions.

Once the model has been compiled, the subsequent step involves fitting the input and output values to the model. This entails training the model on the training dataset using the fit method provided by Keras. During this process, two crucial hyperparameters need to be specified:

1. Epochs: An epoch signifies a complete iteration of the entire training dataset passing through the neural network, encompassing both a forward pass and a backward pass of all training examples. In each epoch, the model generates predictions based on

the input data, computes the error (loss) between the predicted and actual outputs, and updates the network's weights and biases to minimize the loss through backpropagation. This process is repeated for each sample in the dataset.

2. Batch size: Batch size determines the number of samples processed simultaneously by the model during training. It represents the quantity of training examples involved in a single forward/backward pass. For example, if there are 1000 training samples and the batch size is set to 100, the algorithm takes the first 100 samples (from the 1st to the 100th) from the training dataset and trains the network. Subsequently, it proceeds with the second 100 samples (from the 101st to the 200th) and repeats the training process. This iteration continues until all samples have been propagated through the network, completing one epoch, which would take 10 iterations in this case.

When deciding on the values for batch size and epochs, it's important to consider the size of the dataset, the complexity of the model, and the available computing resources. A larger dataset and more complex model may require a larger number of epochs to converge, while a smaller dataset or simpler model may converge quickly with fewer epochs. In general, it's a good practice to start with a smaller batch size and a smaller number of epochs and gradually increase them until the model converges and stops improving.

In summary, at the beginning of the training process, the network's parameters (weights and biases) are initialized with random values, but these are not optimal values that give the minimum error or loss. That's why training process further continues. At every training step (iteration) of the network, the results calculated in the forward propagation step are compared with the ground truth (actual) values to calculate the error or loss score. Then, that error is backpropagated in the backpropagation step to adjust the initial values of the weights and biases so that the error is minimized in the next steps of training. The training is continued until the minimum error is achieved. To compute the error between the predicted values and ground truth (actual) values at every training step, a loss function is required. During the training, the goal is to minimize the loss function as much as possible. To minimize the loss function, an optimizer is needed (optimization algorithm). During the optimization process, the algorithm iteratively takes small steps to minimize the error. These steps possess two key characteristics: direction and size. The direction of each step is determined by the gradient, which is obtained by calculating the derivative of the error

with respect to the weight values. On the other hand, the step size is determined by the learning rate, which governs the speed at which the optimizer descends the error curve. A large learning rate causes the optimizer to take bigger steps, resulting in a faster descent down the error mountain.

In neural networks, after training the model, there are several metrics [30] used to evaluate the performance of the model. The choice of metric depends on the problem and the specific requirements of the application. Here are some common metrics used in neural networks:

1. Precision: Precision measures the proportion of true positives (correct predictions) out of all positive predictions. It is calculated as the ratio of the number of true positives to the total number of positive predictions.

2. Recall: Recall measures the proportion of true positives (correct predictions) out of all actual positive samples. It is calculated as the ratio of the number of true positives to the total number of actual positive samples.

3. F1 score: F1 score is the harmonic mean of precision and recall. It is calculated as: (2 × (precision × recall) / (precision + recall)).

4. The AUC-ROC (Area Under the Receiver Operating Characteristic Curve) is a widely used metric for assessing the performance of classification models, including those based on neural networks. It offers a comprehensive evaluation of a neural network's performance in binary classification tasks, surpassing the limited scope of accuracy alone. The ROC curve is generated by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR) at various classification thresholds. TPR represents the proportion of correctly classified positive samples, while FPR represents the proportion of falsely classified negative samples. The AUC-ROC metric measures the area under the ROC curve, ranging from 0 to 1. A perfect classifier achieves an AUC-ROC of 1, while a random classifier scores 0.5. A higher AUC-ROC indicates superior performance in distinguishing between positive and negative samples. In neural networks, the AUC-ROC is calculated using predicted probabilities for each class to construct the ROC curve. By adjusting the classification threshold, the trade-off between TPR and FPR can be optimized.

Training and evaluating a neural network model involve several key steps. Firstly, the data is prepared by preprocessing, cleaning, and splitting it into training and test sets. Normalization or standardization may also be applied to the input data. Next, the model architecture is defined, including appropriate layers, activation functions, and optimization algorithms. The model is then compiled with a loss function, optimizer, and evaluation metrics. The model is trained using the fit method, specifying the training dataset, epochs, batch size, and other hyperparameters. During training, the model's performance is monitored on a validation set to detect overfitting. After training, the model is evaluated on the testing set to assess its generalization performance using evaluation metrics. Hyperparameters can be tuned to enhance the model's performance. Finally, the trained model can be used to make predictions on new data.

3.3 Autoencoders

Autoencoders (AE) [50] are a type of neural network used for unsupervised learning. They are used to learn a compressed representation of the input data, without the need for labelled training data. The typical architecture of an autoencoder consists of an encoder and a decoder. The encoder takes the input data and compresses it into a lower-dimensional representation, often referred to as the latent space. The decoder then takes this latent representation and reconstructs the original input data. The autoencoder is trained to minimize the reconstruction error between the original input data and the reconstructed data. The learning process of unsupervised autoencoders involves minimizing the difference between the input data and the output data, which is measured by a loss function. The AE is trained using backpropagation, which is an optimization algorithm used to adjust the weights and biases of the neural network to minimize the loss function. Autoencoders have several applications, including image compression, anomaly detection, and data denoising.

Two types of autoencoders are used in this study, fully connected autoencoder and Multi-Layer Perceptron (MLP) autoencoder [51]. A fully connected AE is a type of NN used for unsupervised learning, consisting of fully connected layers. It consists of two parts: an encoder and a decoder. The encoder takes the input data and maps it to a lower-dimensional representation. The decoder then maps the encoded data back to the original input space. The encoding and decoding steps are connected through the intermediate hidden layers. The number of neurons in the intermediate layers are usually smaller than the number of

neurons in the input and output layers. An example architecture of a fully connected autoencoder consists of three layers: an input layer with n input neurons, a hidden layer with m neurons, and an output layer with n input neurons. In this architecture, the input layer has n neurons, and the output layer also has n neurons. The hidden layer has m neurons. The number of neurons in the bottleneck layer, which in this case same as the hidden layer, determines the dimensionality of the compressed representation of the input data. During training, the fully connected autoencoder learns to identify important features and patterns in the input data by trying to reconstruct the input data with minimal error. The autoencoder is trained to minimize the difference between the input data and the output data, which is measured by a loss function such as MSE. Fully connected autoencoders have several applications, for this work, it's used for certifying the runs into either good or bad. This is achieved by computing the loss between the actual data values and the predicted data values generated by the model.

An unsupervised MLP autoencoder is a type of neural network that is used for unsupervised learning and is composed of multiple layers of perceptron, also known as multi-layer perceptron. It consists of multiple layers of fully connected neurons, and it is trained to learn a compressed representation of the input data, similar to the fully connected autoencoder. However, the MLP AE is more flexible in terms of its architecture, and it can handle more complex input data. Like other autoencoders, the purpose of the unsupervised MLP AE is to learn a compressed representation of the input data, without the need for labelled training data. For unsupervised MLP, the encoder and decoder are typically symmetrical, with the same number of layers and the same number of neurons in each layer. An example architecture of an MLP autoencoder with four hidden layers includes an input layer with n input neurons, followed by four hidden layers: hidden layer 1 with m_1 neurons, hidden layer 2 with m_2 neurons, a bottleneck layer with k neurons, and hidden layer 3 with m_2 neurons. Finally, there is hidden layer 4 with m_1 neurons, and an output layer with n input neurons. In this architecture, the input layer has n neurons, and the output layer also has n neurons. The hidden layers have m_1, m_2, neurons, respectively, and the bottleneck layer has k neurons. The number of neurons in the bottleneck layer determines the dimensionality of the compressed representation of the input data. During training, the MLP autoencoder learns to identify important features and patterns in the input data by trying to reconstruct the input data with minimal error. The autoencoder is trained to minimize the difference between the input data and the output data, which is measured by

a loss function such as MSE and MAE. The MLP architecture allows for more complex and non-linear representations to be learned, making it a powerful tool for unsupervised learning tasks. This is because the MLP allows for the use of non-linear activation functions between the layers, which enables the network to learn more complex representations. MLP autoencoder is used to detect anomalous regions of the detector in the input data. Since the autoencoder is trained to reconstruct the input data with minimal error, inputs that cannot be reconstructed well by the autoencoder can be considered anomalies.

In terms of implementation in TensorFlow, both MLP and fully connected AEs can be created using the dense layer, which represents a fully connected layer. The main difference is in the architecture of the autoencoder - MLP autoencoders typically have multiple hidden layers, whereas fully connected autoencoders may have just one hidden layer, as shown in the examples above, the rest of the code is essentially the same.

Both types of AEs are unsupervised learning algorithms that are designed to learn a compressed representation of the input data. This is achieved by encoding the input data into a lower-dimensional space (latent) and then decoding it back to the original input space. One of their applications in this thesis is anomaly detection, where a threshold is set on the reconstruction loss. Any input data that produces a reconstruction loss above the threshold is flagged as an anomaly or outlier as described in Fig. 3.8. Overall unsupervised autoencoders are a powerful tool for learning compressed representations of data and are used in this thesis for two applications, including certification of runs and anomaly detection.

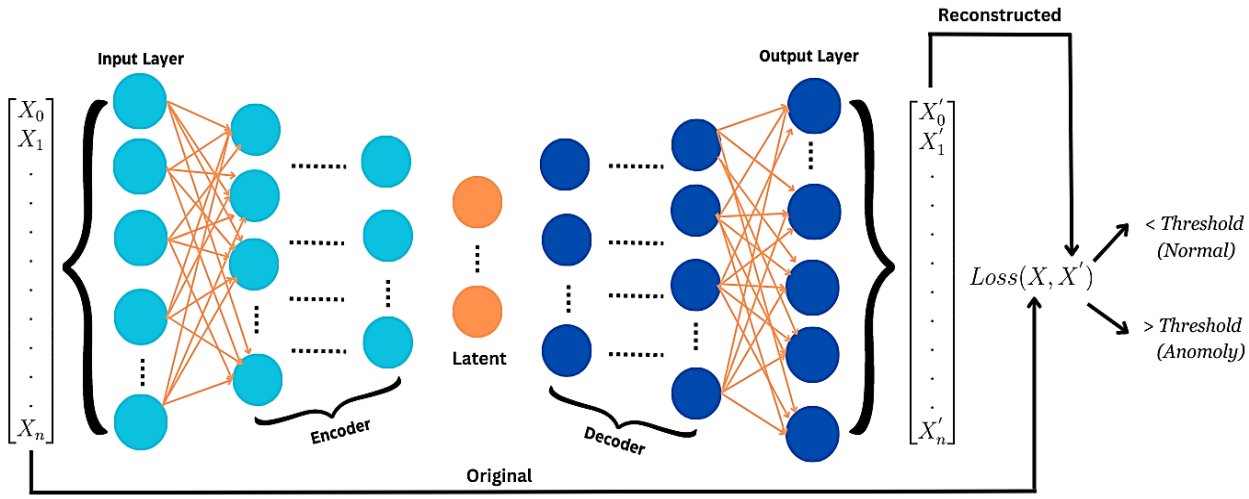


Figure 3.8. Fully connected autoencoder structure.

4. Analysis

This chapter presents the datasets used and the considered methodology.

4.1. Data Sets Description

This thesis focuses on analyzing data obtained from the CMS detector during RUN-2 in 2018 and RUN-3 in 2022, corresponding to center-of-mass energies of 13 TeV and 13.6 TeV, respectively as seen in Fig. 4.1 and Fig. 4.2. The dataset utilized in this analysis has an integrated luminosity of 63.67 fb^{-1} and 38.48 fb^{-1} for each respective run. A detailed view of the data taking overtime is presented in the figures below. The findings demonstrate that the majority of events provided by the LHC are recorded by the CMS detector. To ensure data quality, various criteria must be met for the inclusion of data in this analysis. Only runs with proper functioning of the accelerator and detector are selected, and solely data from the single muon stream is considered and evaluated.

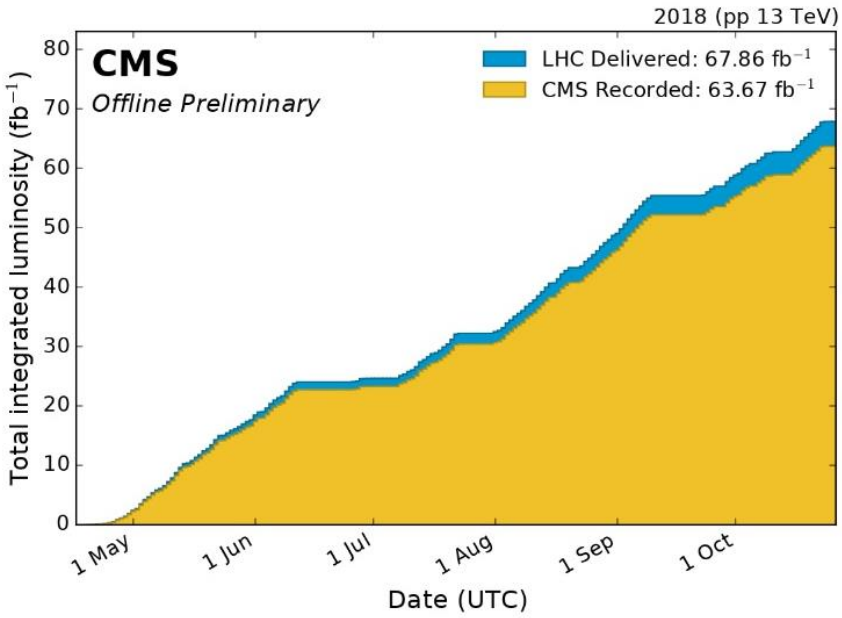


Figure 4.1. These plots depict the measured luminosity delivered by the LHC to CMS (blue) and the luminosity recorded by CMS (orange) during stable beams and proton-proton collisions at a center-of-mass energy of 13 TeV in 2018. The delivered luminosity encompasses the duration from the initiation of stable beams until the beams are dumped by the LHC [52].

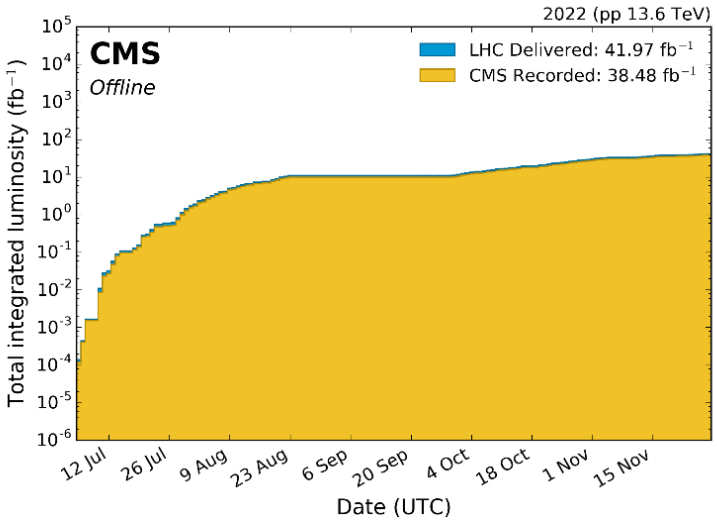


Figure 4.2. These plots illustrate the measured luminosity delivered by the LHC to CMS (depicted in blue) and the luminosity recorded by CMS (depicted in orange) during stable beams and proton-proton collisions at a center-of-mass energy of 13.6 TeV in 2022. The delivered luminosity encompasses the duration from the initiation of stable beams until the beams are dumped by the LHC [52].

In this dissertation, different histograms of monitor elements are used. In addition to the MET histograms, jet monitor elements taken from the CMS DQM GUI Cleanedak4PFJetsCHS are used and described in Table 4.1 below. Also, 2-dimension histograms are used, including one histogram (Phi VS Eta) taken from Cleanedak4PFJetsCHS, and two other histograms from PF Candidates as seen in Table 4.2.

Table 4.1. Jet monitor elements and their description. Cited from internal CMS Twiki.

Jet Monitor Element	**Description**	**#**
CHEn highPt, lowPt, mediumPt (Barrel, EndCap)	The energy of charged hadron from jets of high, low, medium p_T for HB and HE.	6
CHFrac highPt, lowPt, mediumPt (Barrel, EndCap)	Charged hadron energy fraction relative to corrected jet energy of high, low, medium p_T for HB and HE.	6
Charged Multiplicity highPt, lowPt, mediumPt (Barrel, EndCap)	Number of charged particles from jets of high, low, medium p_T for HB and HE.	6
Charged Em Energy, Charged Hadron Energy, Charged Mu Energy Charged Multiplicity	Charged electromagnetic energy, Charged hadron energy, Charged muon energy, and Charged multiplicity (average number of charged particles).	4

Constituents (Barrel, EndCap, Forward)	Number of jet constituents in each HCAL detector.	3
Constituents, Constituents uncor	Number of jet constituents, Uncorrected Constituents (without jet energy corrections).	2
El En highPt, lowPt, mediumPt (Barrel, EndCap)	Electron energy from jets of high p_T for HB and HE.	6
Eta, Eta First, Eta uncor	Pseudo rapidity (the angle of a particle in relation to the particle beam or z-axis) for each of the sub detector components, Eta value of the leading jet, and Uncorrected pseudo rapidity.	3
HFE En Forward (highPt, lowPt, mediumPt)	The jet electron energy in HF for different p_T ranges.	3
HFE Frac Forward (highPt, lowPt, mediumPt)	Electron energy fraction relative to corrected jet energy of high, low, medium p_T in HF.	3
HFH En Forward (highPt, lowPt, mediumPt)	The jet hadronic energy in HF for different p_T ranges.	3
HFH Frac Forward (highPt, lowPt, mediumPt)	Hadron energy fraction relative to corrected jet energy in HF for different p_T ranges.	3
Jet Energy Corr	Corrected jet energy (the ratio of corrected jet p_T and calo jet p_T).	1
Jet Mass highPt, lowPt, mediumPt (Barrel, EndCap, Forward)	The mass of jets of high, low, medium p_T for HB and HE.	9
Mu En highPt, lowPt, mediumPt (Barrel, EndCap)	Muon energy from jets of high, low, medium p_T for HB and HE.	6
Mu Multiplicity highPt, lowPt, mediumPt (Barrel, EndCap), Muon Multiplicity	The number of muons from jets of high, low, medium p_T for HB and HE and multiple muon measurements.	7
NH En highPt, lowPt, mediumPt (Barrel, EndCap)	Neutral hadron energy from jets of high, low, medium p_T for HB and HE.	6
NH Frac highPt, lowPt, mediumPt (Barrel, EndCap)	Neutral hadron energy fraction relative to corrected jet energy of high, low, medium p_T for HB and HE.	6
Neutral Multiplicity highPt, lowPt, mediumPt (Barrel, EndCap, Forward)	The number of neutral particles from jets of high, low, medium p_T for HB and HE.	9
Neutral Constituents Fraction	Neutral constituents' fraction relative to corrected jet energy	1

Neutral Em Energy, Neutral Hadron Energy	Neutral electromagnetic energy from jets, Neutral hadron energy from jets.	2
Neutral Multiplicity	The total number of neutral particles from jets.	1
Neutral Em Energy, Neutral Hadron Energy	Neutral electromagnetic energy from jets, Neutral hadron energy from jets.	2
Neutral Multiplicity	The total number of neutral particles from jets.	1
N jets	Number of jets.	1
Ph En highPt, lowPt, mediumPt (Barrel, EndCap)	Photon energy from jets of high, low, medium p_T for HCAL Barrel and EndCap.	6
Ph Frac highPt, lowPt, mediumPt (Barrel, EndCap)	Photon energy fraction relative to corrected jet energy of high, low, medium p_T for HB and HE.	6
Phi	The total measurement of the azimuthal angle for corrected jet.	1
Phi (Barrel, EndCap, Forward)	Phi for corrected jet in each detector.	3
Phi First, Phi uncor	Phi value of the leading jet and phi of uncorrected jet.	2
Pt	Total transverse momentum of corrected jet.	1
Pt (Barrel, EndCap, Forward)	p_T of corrected jets for each of the HCAL detectors.	3
Pt (1,2,3)	p_T of corrected jet for different ranges (0-100, 0-300, 0-5000).	3
Pt First, Pt uncor	p_T value of the leading jet and p_t of uncorrected jet.	2

Table 4.2. Monitor histograms and their description. Cited from internal CMS Twiki.

Monitor Histogram	**Description**
Phi VS Eta	Two-dimension histogram of eta and phi of the leading jet.
Hadron Occupancy	Two-dimension histogram of eta and phi of particle flow hadron candidates.
Neutral Hadron EndCap Occupancy	Two-dimension histogram of eta and phi of particle flow neutral hadron candidates in Endcap.

The dataset used in this study consists of runs from the 2018 prompt-reco, specifically the JetHT dataset, which is effective for jet variables and uses jet triggers to collect data, resulting in more jets than other datasets. There are 518 runs in the 2018 prompt-reco dataset, but 31 runs were excluded for two reasons. Firstly, those runs were from different datasets, such as ZeroBias and StreamExpress. Secondly, some of the runs suffered from timing scan problems and low statistics, which could potentially bias the ML models. Therefore, the dataset used in the study contains 487 runs, with 465 runs declared of good quality based on the "Golden Json" file of the CMS, and 22 runs declared as bad according to the CMS Run Registry JME group.

4.2. Methodology

This work focuses on using machine learning methods and algorithms in order to evaluate the CMS offline DQM runs. There are two different matters that will be presented within this thesis. First, certification of runs will be evaluated with supervised and unsupervised machine learning. Then, different autoencoder models will be trained on Jet and MET histograms to detect anomalous regions of the CMS detector.

The structure of this study is divided into two parts, supervised and unsupervised. Starting with the supervised approach, gradient descent method is used to determine which observables are the most effective for data certification. After that, different supervised classifiers are trained on the variables selected using the gradient descent approach. for the second part, unsupervised autoencoder is trained on a set of jet observables for the aim of certifying the runs. Finally, to identify anomalous regions of the detector, different autoencoders are trained on MET histogram and two-dimension histograms such as phi-eta histograms and occupancy plots.

5. Results

This chapter presents the results of training different machine learning models on CMS DQM datasets to automate the offline data certification procedure.

5.1. Supervised Learning

In order to determine which jet observable provides the most power in identifying anomalies, supervised models are trained on a set of already certified runs. By using a

labeled dataset of certified runs, the supervised models can learn patterns and correlations between the different observables and the occurrence of anomalies.

5.1.1. Choice of Input Features: Feature Reduction

This study focuses on the development and training of a fully connected neural network classifier using the 2018 prompt-reco data. The primary aim is to minimize the number of features prior to implementing autoencoders. Given our focus on encoding the features that are crucial for differentiating between good and bad runs, it is crucial to prioritize feature reduction. This step is particularly important because unsupervised algorithms, including autoencoders, lack the understanding of feature importance and may attempt to encode all features indiscriminately. To achieve this, a binary classification task is performed, where each run is assigned a label of either 0 (good) or 1 (bad). The input data for the classifier consists of the mean values of the 124-jet observable distributions for each run. The training of the neural network classifier is carried out using the binary cross entropy loss function, which is suitable for this type of classification problem.
To ensure that the mean values fall within the range of 0 and 1, the dataset is normalized. Then, the dataset is split into training and testing sets using the Scikit-learn function "train_test_split," with a split ratio of 70% for training and 30% for testing. This division ensures that the model is trained on a sufficiently large portion of the data while retaining a separate subset for evaluation purposes.

The model architecture consists of an input layer followed by batch normalization, two dense layers with SELU activation, and dropout layers. The input layer takes runs with a shape of 124 features each. Batch normalization is applied to normalize the activations of the previous layer, aiding in training stability and speed. The dense layers with 90 and 30 nodes respectively utilize the SELU activation function to address gradient vanishing issues. Dropout layers with a rate of 0.1 are inserted after each dense layer to prevent overfitting. The output layer employs a sigmoid activation for binary classification, producing the model's predictions. The model is compiled with the Adam optimizer and a learning rate of 0.0001. Binary cross-entropy loss is used for training, and binary accuracy is chosen as the evaluation metric with a threshold of 0.5. Subsequently, the model is fitted to the training instances using a batch size of 15 and trained for 100 epochs. The choice to limit the training to 100 epochs aims to prevent overfitting and find a balance between model performance and training time.

After the training phase, predictions are made on the testing instances using the learned parameters from the fit. The reconstructed outputs are then evaluated using the loss function and other relevant evaluation metrics to assess the performance of the classifier.

We used various metrics, including train and test accuracy, train, and test loss, as well as ROC AUC scores. The obtained results are as follows: Test loss = 0.03142, Test accuracy = 0.9858, Train loss = 0.05894, Train accuracy = 0.9674, ROC AUC score of training set = 0.999, ROC AUC score of test set = 1.0.

These metrics provide insights into the model's performance on both the training and testing datasets. The test loss value indicates the average loss of the model's predictions on the testing dataset, with a lower value indicating better performance. The test accuracy represents the percentage of correctly classified instances in the testing dataset. Similarly, the train loss and train accuracy metrics measure the model's performance on the training dataset. A lower train loss and a higher train accuracy indicate that the model has successfully learned the patterns and features in the training data. Overall, the reported metrics demonstrate promising results, with high accuracy, low loss values, and excellent ROC AUC scores on both the training and testing datasets as seen in Figures 5.1, 5.2, and 5.3. These outcomes indicate that the model has learned the relevant patterns and features to effectively classify the runs as either good or bad.

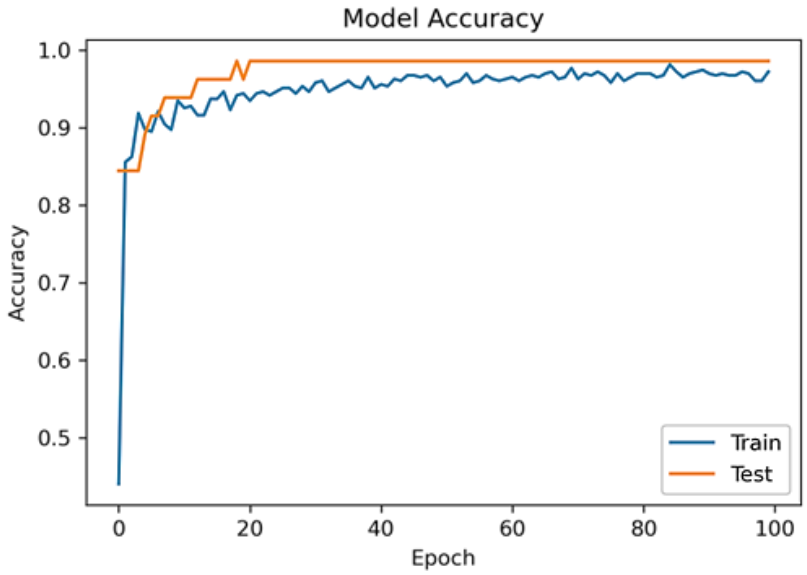


Figure 5.1: Epoch-wise Accuracy of the model during training and testing. The x-axis represents the epoch number, while the y-axis represents the accuracy of the model.

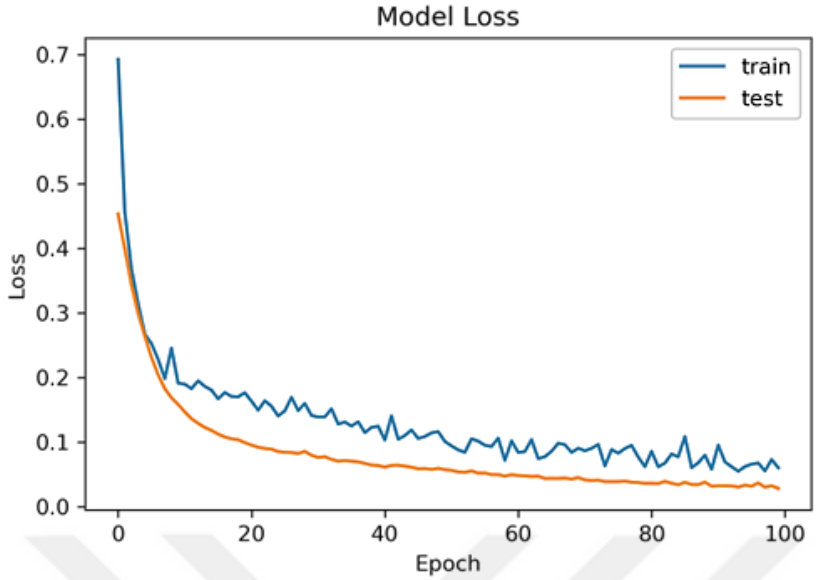


Figure 5.2: Loss of the model during training and test epochs. The x-axis represents the epoch number, while the y-axis represents the loss value.

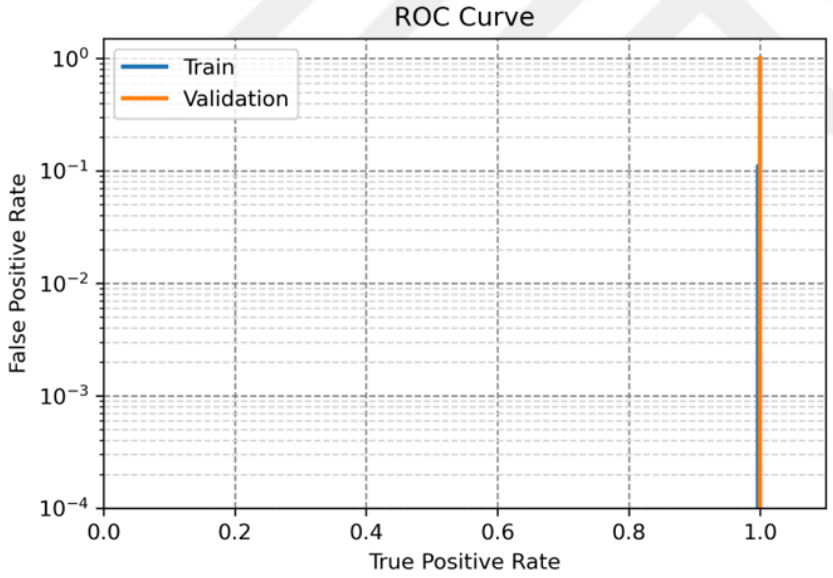


Figure 5.3: The ROC curve for training and test sets.

To optimize the selection of input features, we perform a variable reduction method using gradient descent. The first order gradient of the loss is calculated with respect to the input features to determine their sensitivity to the predictions. However, to mitigate the effect of noise in the gradient descent-based feature reduction method, we adopted an ensemble-based approach. In this approach, 25 separate models would be trained, each with its own set of distinct initial conditions. This involves randomly initializing the weights and biases

for each model. By utilizing a range of different initial conditions, the ensemble captures diverse regions of the parameter space and helps to avoid potential local optima, leading to improved overall performance and robustness. Then, the average values of the first order gradient for each jet variable are calculated across all models and the results are seen in Fig. 5.4 and Fig. 5.5 below.

The analysis reveals that the variables with the highest first order gradients are predominantly the jet energy fractions. These fractions provide information about the distribution of energy among the particles within a jet. This indicates that jet energy fractions play a significant role in differentiating between good and bad runs and are the most influential variables for DC. This conclusion is considered 100% correct and has been validated by the expertise of DQM specialists.

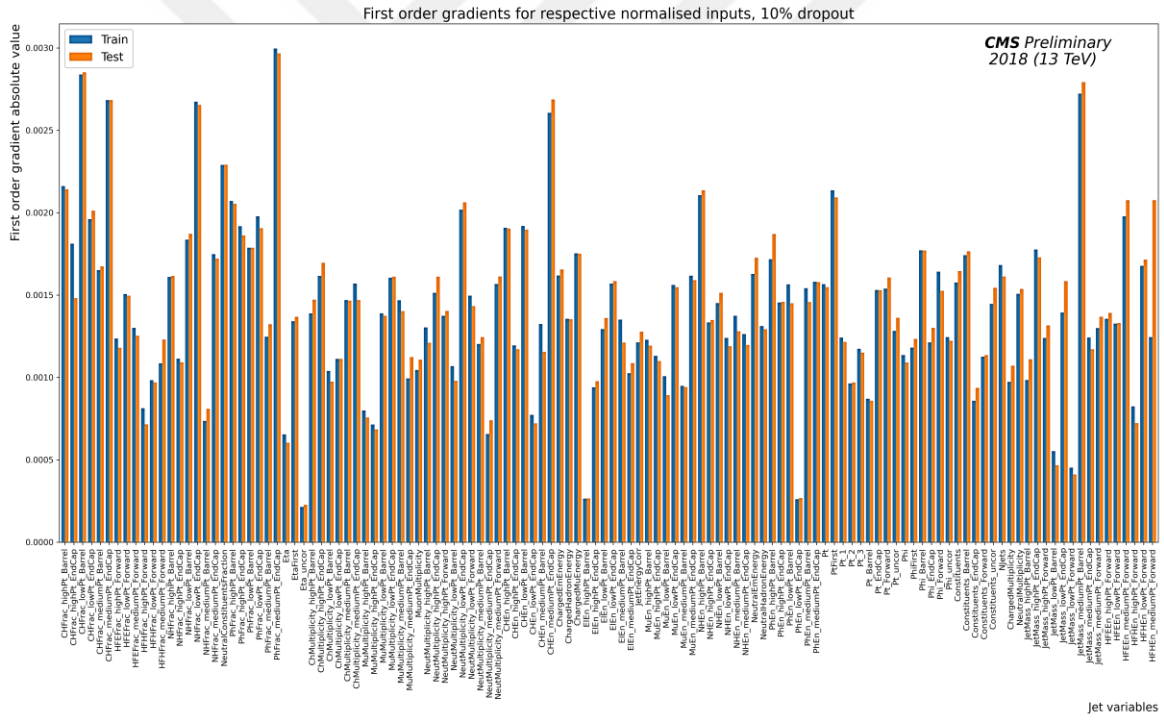


Figure 5.4: The absolute values of the first order gradients for all 124 variables for both training and test instances.

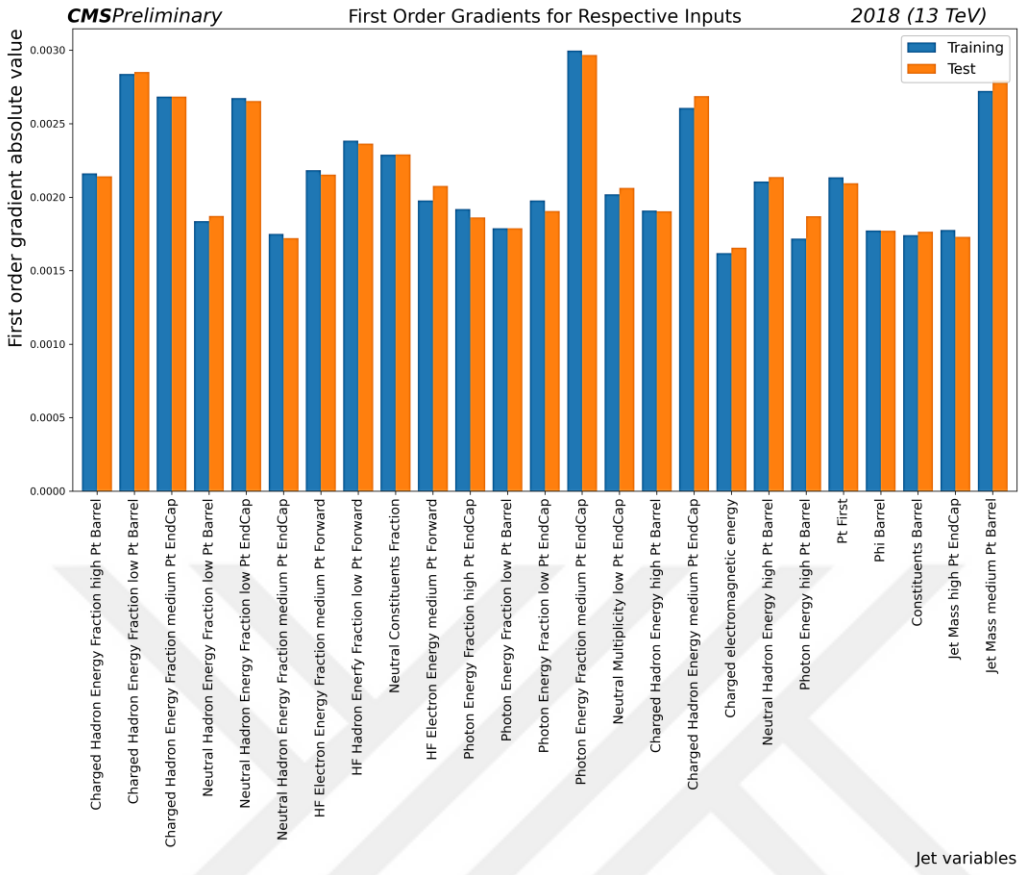


Figure 5.5: The absolute values of the first order gradients for the first 25 features with the highest gradients values for both training and test instances.

5.1.2. DC with Classification Algorithms

In the supervised approach, the next step involves training those jet energy fraction observables using different classifiers. The goal is to assess the effectiveness of these observables in detecting potentially anomalous runs. To ensure the suitability of jet energy fractions in accurately classifying the runs, we test with three different classifiers:

1. K-Nearest Neighbours (KNN)
2. Gaussian Naive Bayes (Gaussian-NB)
3. Support Vector Machine (SVM)

The dataset used for training and testing includes the mean values of 25 jet fractions. These fractions were selected using the feature reduction method and encompass various observables such as Charged Hadron Energy Fraction, Neutral Hadron Energy Fraction, Photon Energy Fraction, Hadron Energy, and Electron Energy Fraction for the Forward

region. Additionally, the Neutral Constituents Fraction is considered for both the Barrel and EndCap regions, covering low, medium, and high p_T ranges.

To classify the runs, the probability distributions are examined (see Figures 5.6, 5.7, and 5.8). If the probability assigned to a run is less than 0.5, it is classified as bad, indicating a potentially anomalous run. Conversely, if the probability exceeds or equals 0.5, the run is classified as good. The figures provided depict the probability distributions for each classifier, both for the training and test sets, enabling an evaluation of their classification performance.

By employing these classifiers and utilizing the selected jet energy fraction observables, the aim is to gauge their discriminatory power in detecting anomalous runs and verify their effectiveness in accurately classifying the runs.

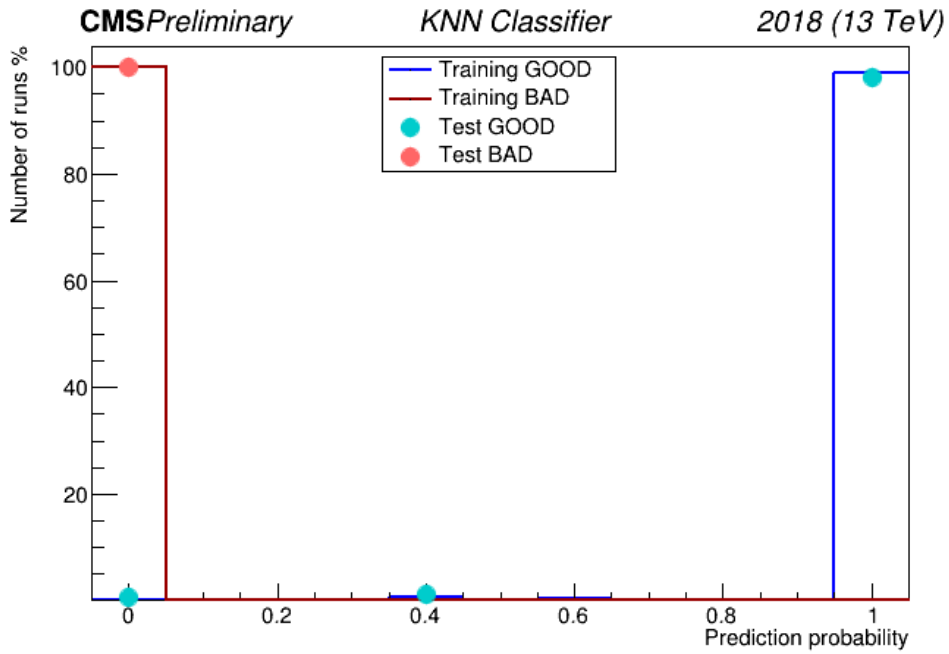


Figure 5.6. Probability distribution of KNN Classifier Predictions. The x-axis represents the predicted probability of runs, while the y-axis represents the percentage of runs corresponding to each probability range.

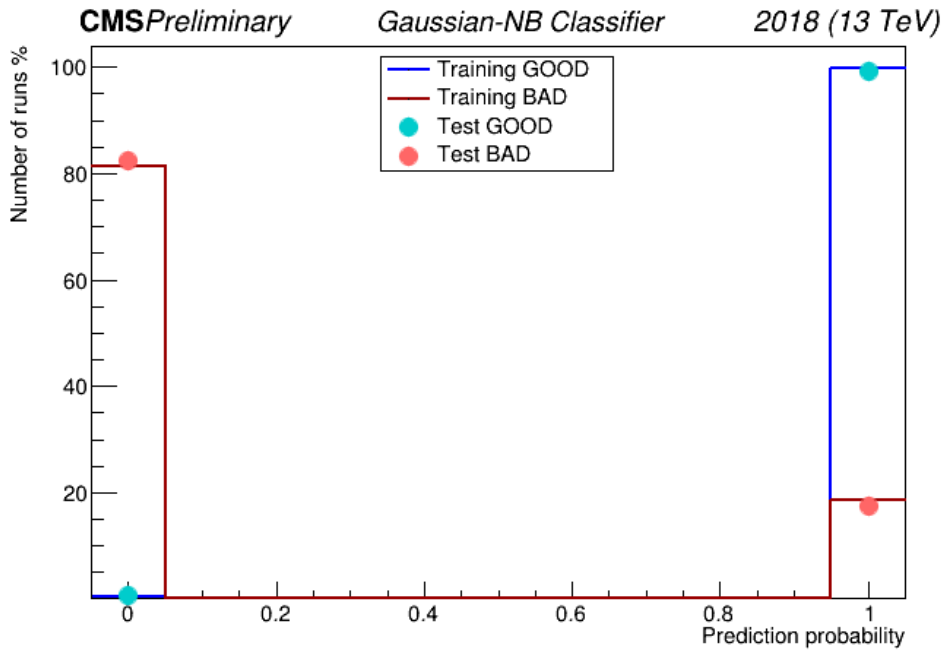


Figure 5.7. Probability Distribution of Gaussian-NB Classifier Predictions. The x-axis represents the predicted probability of runs, while the y-axis represents the percentage of runs corresponding to each probability range.

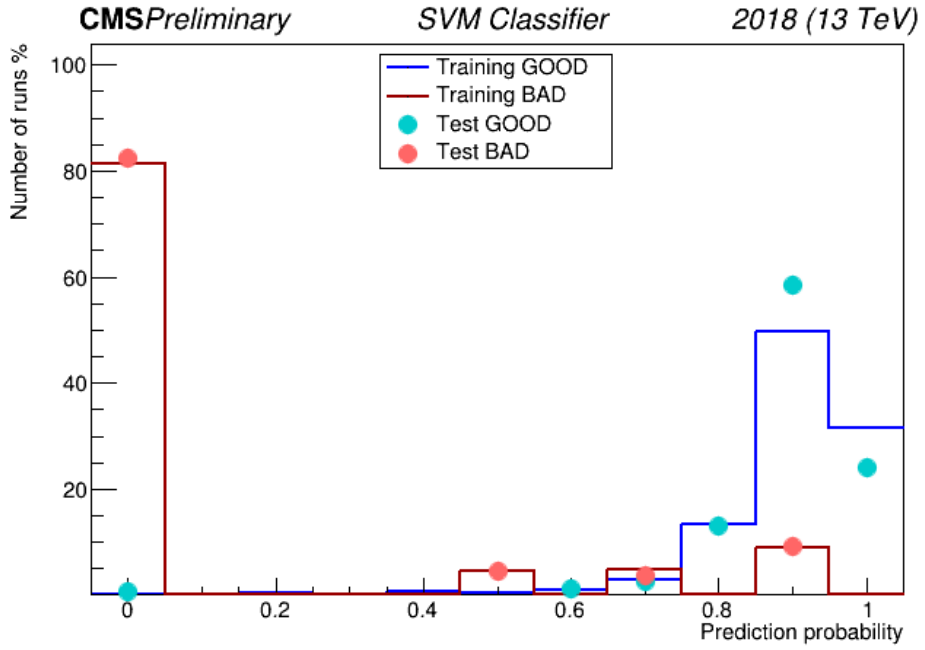


Figure 5.8. Probability Distribution of SVM Classifier Predictions. The x-axis represents the predicted probability of runs, while the y-axis represents the percentage of runs corresponding to each probability range.

The evaluation of the three classifiers employed in this study has demonstrated their exceptional performance in certifying the runs, as evidenced by their ROC AUC scores shown in the Figures 5.9, 5.10, and 5.11 below. Notably, both the training and test sets have

exhibited remarkable results, surpassing a ROC AUC score of 0.90. The performance metrics for each classifier are as follows:

KNN: roc_auc_score_train = 1.0, roc_auc_score_test = 0.9967

GNB: roc_auc_score_train = 0.92949, roc_auc_score_test = 0.99675

SVM: roc_auc_score_train = 0.9552, roc_auc_score_test = 0.95276

Among these classifiers, KNN has emerged as the top performer, exhibiting the highest ROC AUC scores. This implies that KNN has shown exceptional capability in accurately distinguishing between good and bad runs, surpassing the performance of GNB and SVM.

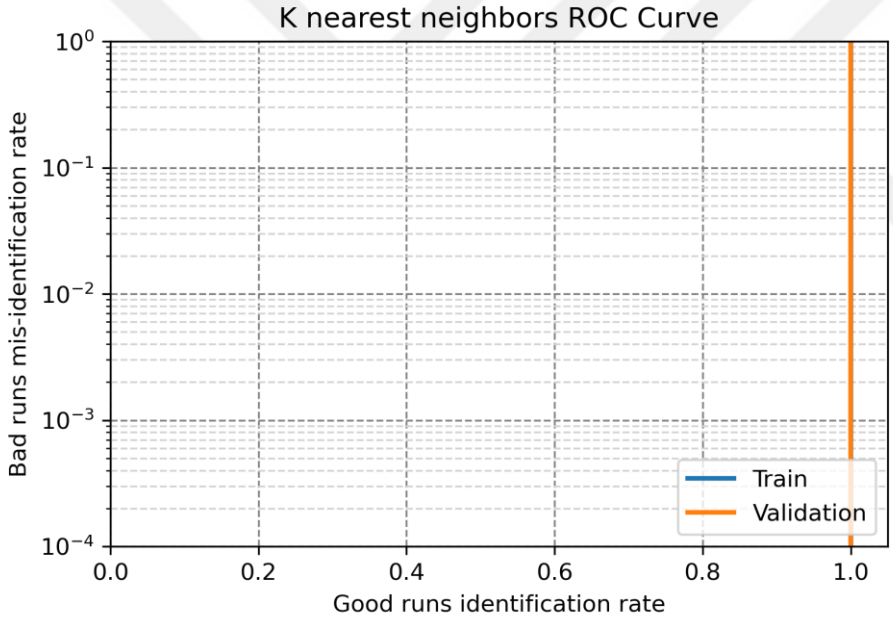


Figure 5.9. The ROC Curve for the KNN classifier.

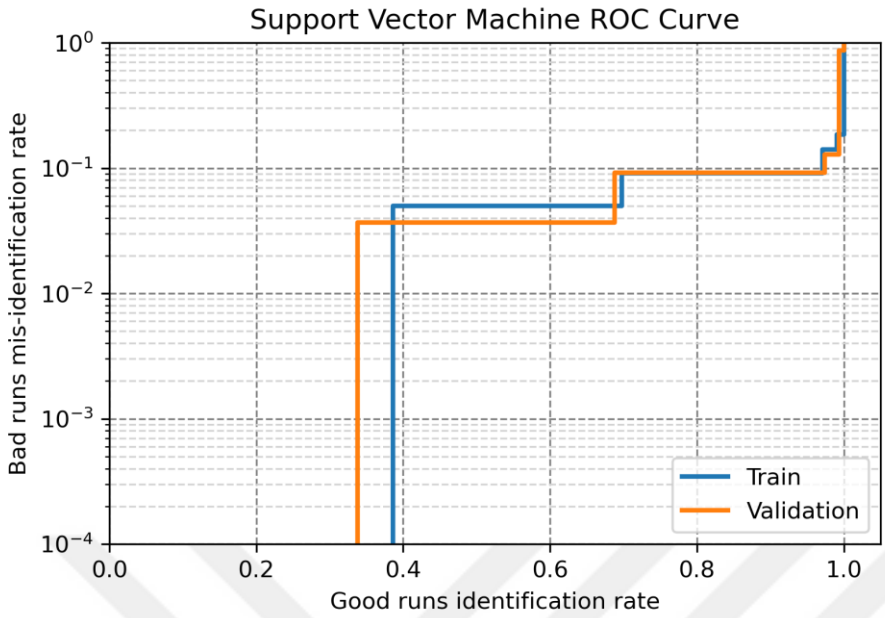


Figure 5.10. The ROC Curve for the Gaussian-NB classifier.

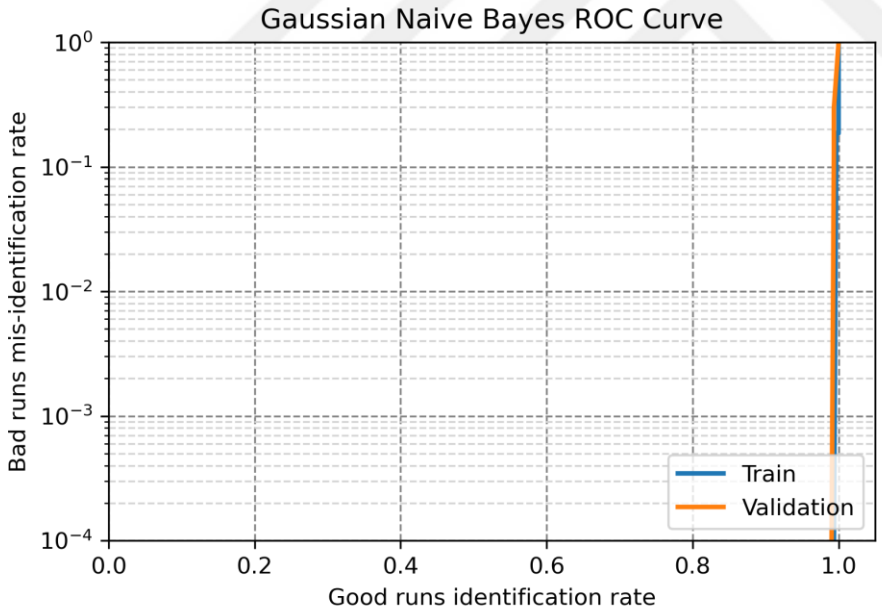


Figure 5.11. The ROC Curve for the SVM classifier.

5.2. Unsupervised Learning

Moving to the next part of the study where we used unsupervised autoencoders to certify the runs and detect anomalous regions of the detector.

5.2.1. DC with Autoencoders

In this section, we present our approach for certifying runs by leveraging the impact of jet fraction features and training them with an autoencoder. The proposed approach entails training the autoencoder on good runs only and subsequently evaluating its performance on a combination of good and bad runs.

To train the autoencoder, we employed the MSE as the loss function. The process began by identifying the highest MSE value within the training dataset of good runs. This value is then utilized as a threshold for subsequent classifications. By applying this threshold to the maximum MSE values of the test dataset, we established a cut-off point for run classification. Any run with a maximum MSE value exceeding the determined threshold is classified as bad, while those below the threshold are classified as good.

The fully connected AE is implemented using TensorFlow's Keras API. It is created by first defining an input layer with a shape of 25 for each run. Then, the encoder which consists of a series of dense layers is added, with 100 and 50 nodes respectively, using the SELU activation function. Dropout layers with a dropout rate of 0.1 are applied after each dense layer to prevent overfitting. The decoder or the output layer consists of a dense layer with the same number of nodes as the input layer, using the sigmoid activation function. The model is compiled using the Adam optimizer with a learning rate of 0.001 and the MSE loss function. The training process is executed for 100 epochs with a batch size of 10.

Three distinct datasets were employed in this study to assess the effectiveness of our approach. Firstly, we trained and tested the model using runs from Run-2 2018. Then, we expanded our training dataset to include runs from both Run-2 2018 and Run-3 2022, while testing with different runs from Run-3 2022. So, in the second and third datasets, we incorporated runs from various Cycles conducted during the 2022 Run-3 phase. Each Cycle represents a distinct collection of runs that occurred at different data-taking periods.

The three datasets demonstrated high performance with an accuracy exceeding 90% in certifying the runs. The results are presented for each dataset below.

1) Train and test with runs from Run-2 2018.

The following Fig. 5.12 illustrates MSE of the 2018 training good runs for each of the 25 features. The maximum MSE value observed in the training good runs, set at 0.00075, serves

as the threshold for the test runs. To gain a better understanding of the methodology, an example of one specific good run, "316876", from the 2018 training runs is presented. Fig. 5.13 (a) showcases the distribution of the input and reconstructed mean values, while Fig. 5.13 (b) displays the MSE for this particular run across the 25 features.

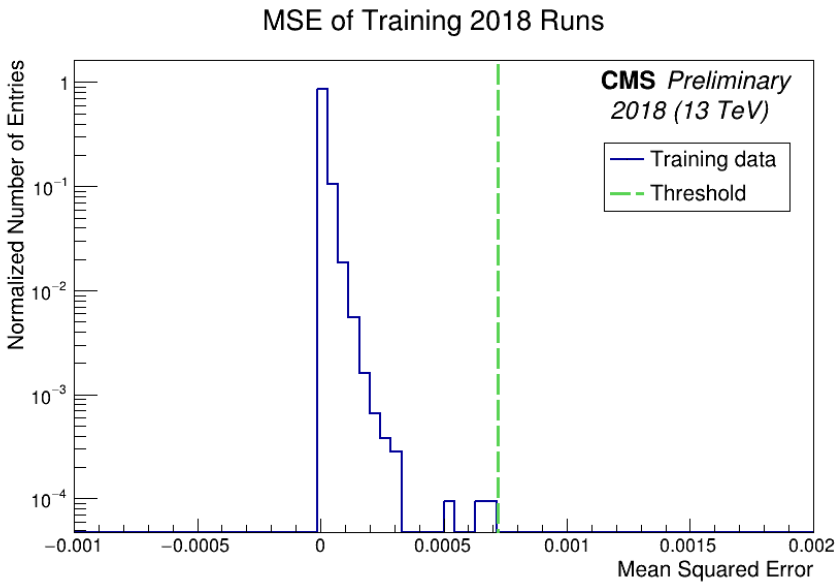
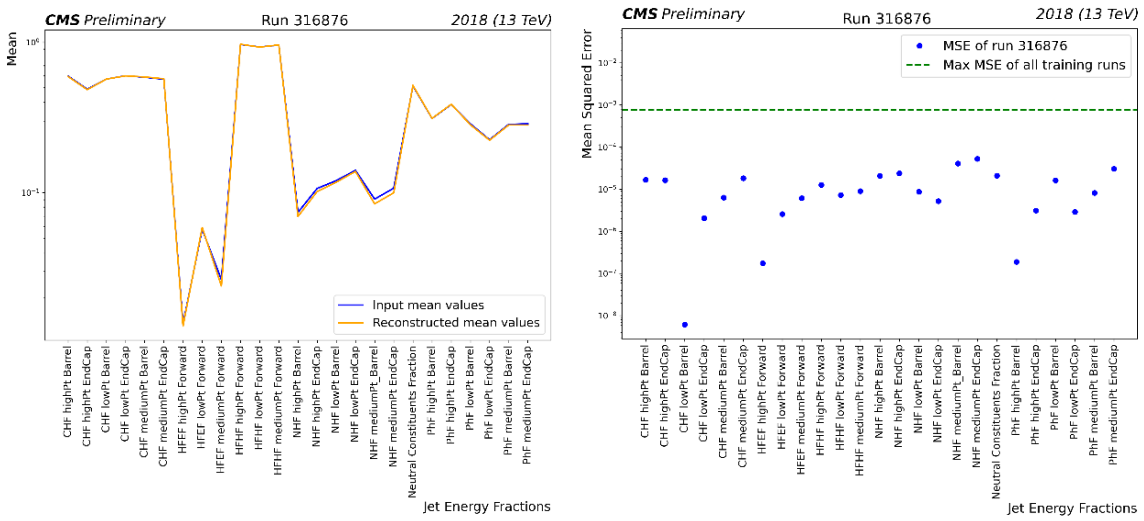


Figure 5.12. The mean squared error of 2018 training good runs for each of the 25 features and the chosen threshold for test runs.



(a) (b)

Figure 5.13. (a) displays the original mean values (represented by the blue curve) of the 25 jet fraction variables for the good run, while the orange curve illustrates the reconstructed mean values. In (b), the blue dots indicate the MSE for the good run. The threshold is determined based on the highest MSE observed during training of the good runs, and it is set at 0.00075.

After training, we test with different good and bad runs from 2018 as shown in Fig. 5.14 and Fig. 5.15. With this chosen threshold we were able to correctly certify all test good runs, demonstrating a perfect certification rate for this class. However, a small portion, specifically 3 runs, of the test bad runs were misclassified. These misclassified runs account for approximately 7% of the total runs in the test set.

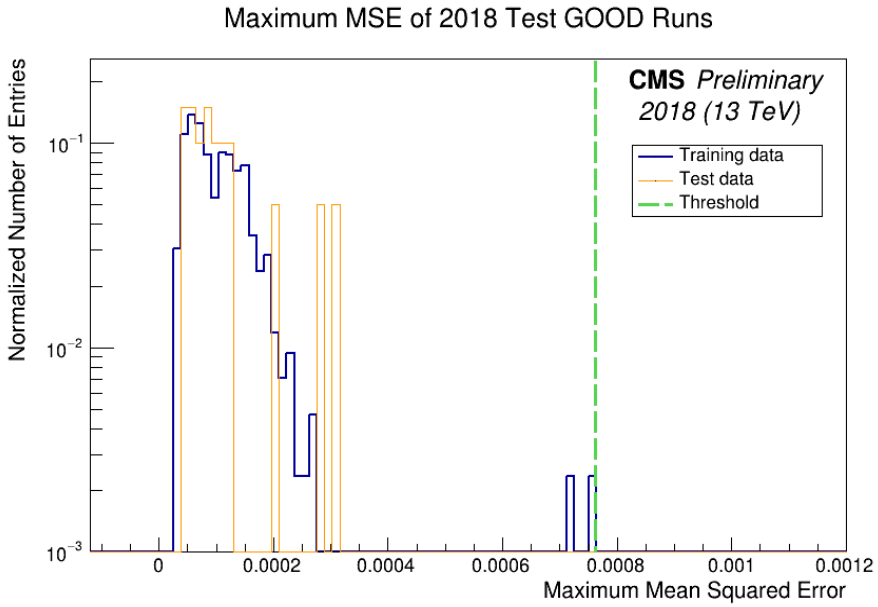


Figure 5.14. Distributions of the max MSE for training good runs in blue and test good runs in orange.

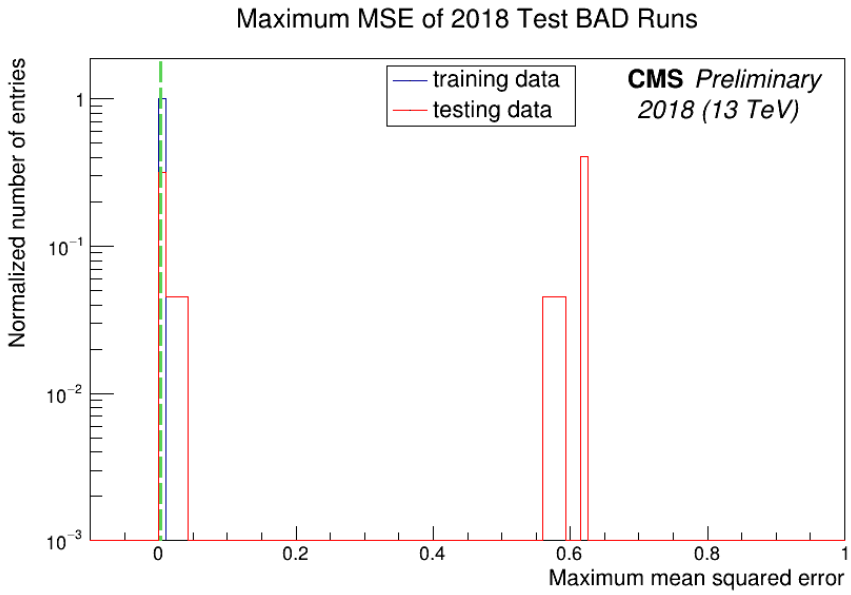


Figure 5.15. Distributions of the max MSE for training good runs in blue and test bad runs in red.

Following the certification process, we proceeded to calculate the ROC AUC scores for both the training and test sets. The ROC AUC score for the training runs is 1, while the ROC AUC score for the test runs is found to be 0.93, indicating a great classification performance on both sets. These scores demonstrate the effectiveness of the model in accurately classifying the runs. To visualize the model's performance, we generated the ROC curve, presented in Fig. 5.16. This curve provides a graphical representation of the classification performance on both the training and test runs. The x-axis represents the identification rate of good runs, while the y-axis represents the misidentification rate of bad runs. The ROC curve further reinforces the model's ability to accurately classify runs across the chosen threshold, enabling a comprehensive evaluation of its performance.

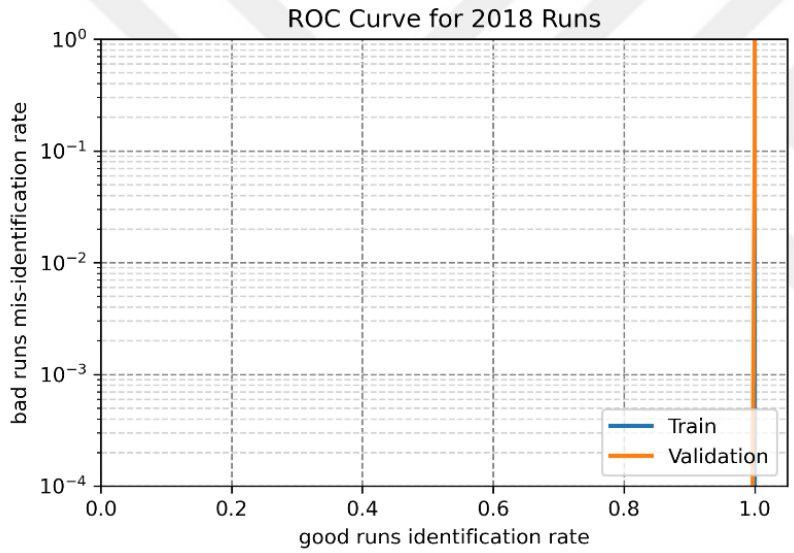


Figure 5.16. ROC curve of binary classification of 2018 runs in training and test/ validation sets.

2) Train with runs from Run-2 2018 and Run-3 2022 Cycle 33, and test with different runs from Run-3 2022 Cycles 33 and 34.

Figure 5.17 presents the MSE of training good runs from Run-2 2018 and Run-3 2022 Cycle 33, along with the chosen threshold for test runs of Cycle 33 and Cycle 34. To ensure accurate measurements, we trained the AE multiple times with different initial conditions and averaged the maximum MSE values across all models. The MSE threshold was then tuned to minimize misclassification of bad runs. However, with a threshold of 0.002, one good run from the training set exceeds the threshold and is misclassified. Fig. 5.18 and Fig. 5.19 display the maximum MSE of each run for training and test instances, respectively.

Notably, with this threshold, all test good runs and test bad runs are correctly classified, achieving a 100% classification rate.

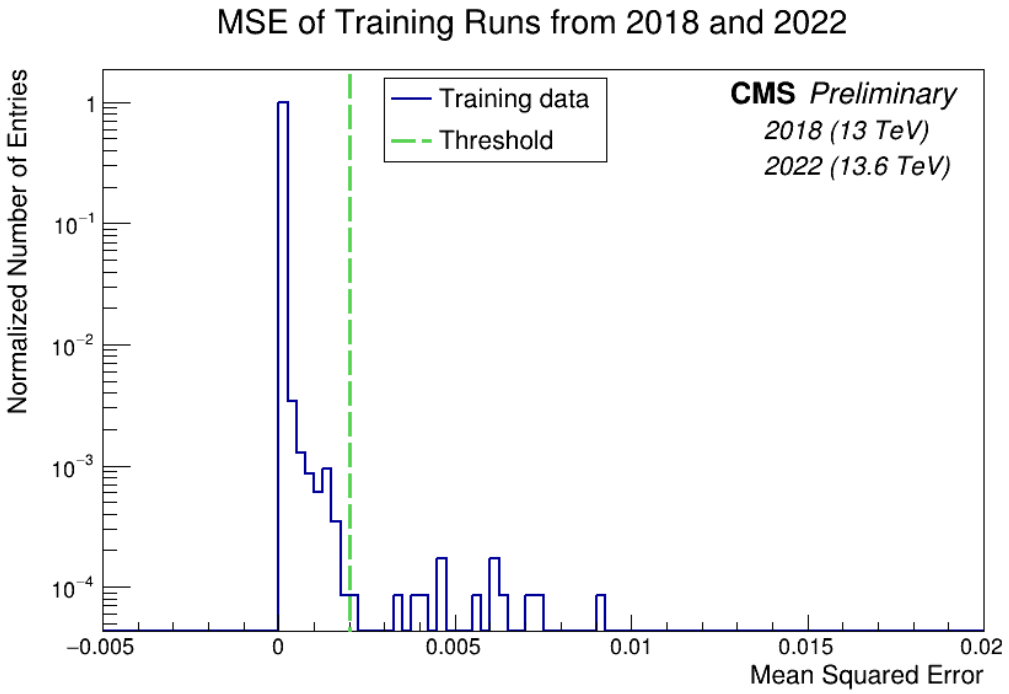


Figure 5.17. The mean squared error of 2018 and 2022 Cycle 33 training good runs for each of the 25 features and the chosen threshold for test runs.

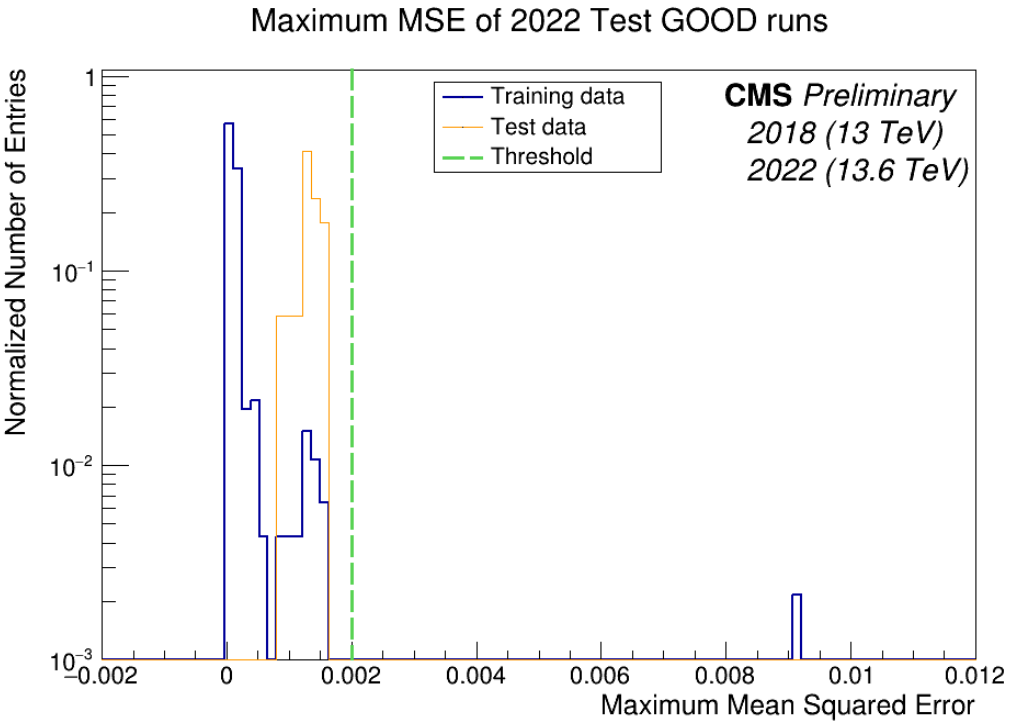


Figure 5.18. Distribution of the max MSE for training good runs in blue and test good runs in orange.

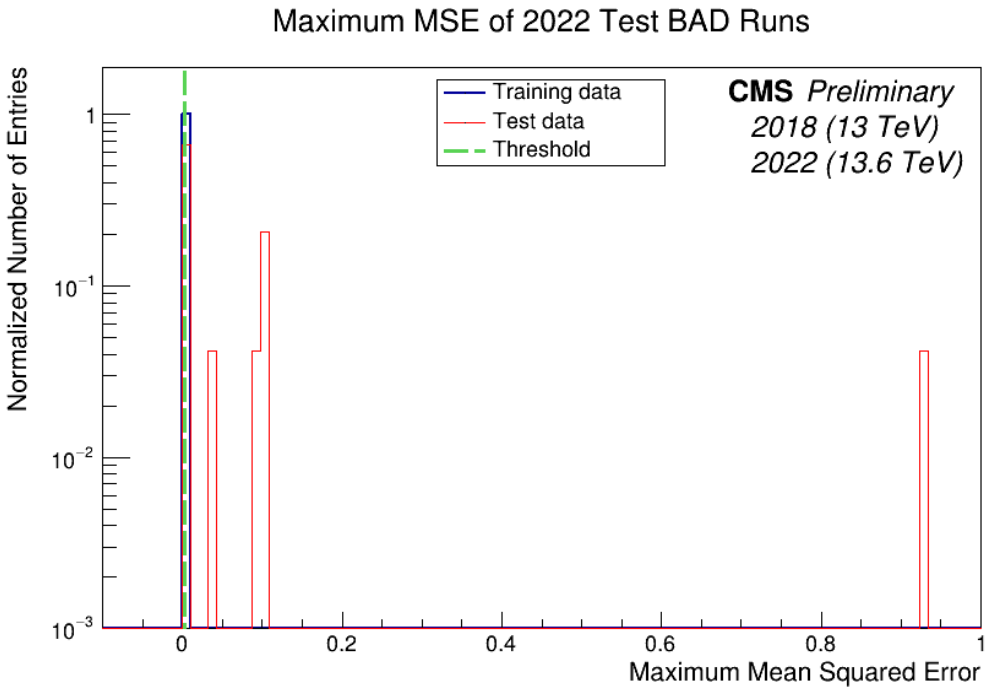


Figure 5.19. Distribution of the max MSE for training good runs in blue and test bad runs in red.

Following the calculation of ROC AUC scores, we obtained a score of 0.998 for the training runs and a perfect score of 1 for the test runs. These scores reflect the exceptional classification performance of the model on both sets (see Fig. 5.20).

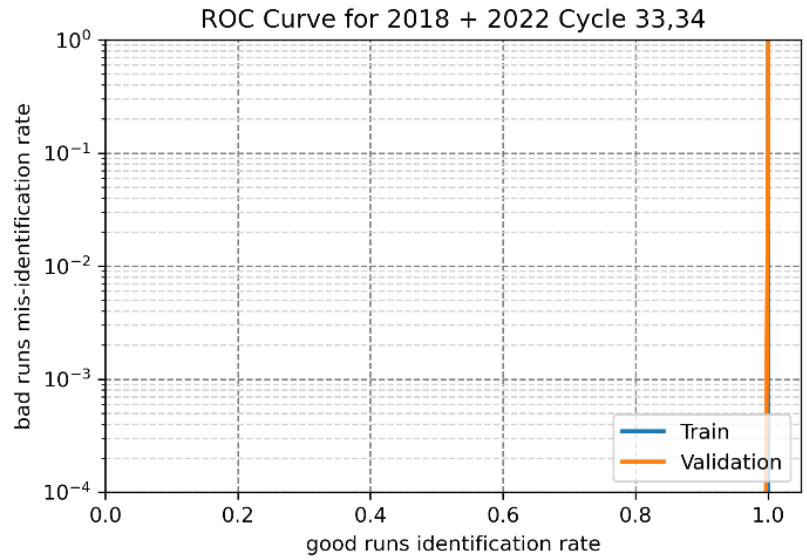


Figure 5.20. ROC curve of binary classification of 2018 and 2022 runs in training and test or validation sets.

3) Train with runs from Run-2 2018 and Run-3 2022 Cycle 33 and Cycle 39, and test with runs from Run-3 2022 Cycle 39 and Cycle 40.

Figure 5.21 presents the MSE of training good runs from Run-2 2018 and Run-3 2022 Cycle 33 and Cycle 39, along with the chosen threshold for test runs of Cycle 39 and Cycle 40. Similar to our approach in the previous dataset, we employed multiple training iterations of the AE with varying initial conditions to obtain accurate measurements. The maximum MSE values were averaged across all models, and the MSE threshold was fine-tuned to minimize misclassification of bad runs. However, with a threshold of 0.0025, it is one good run from the training set exceeds the threshold and is misclassified. This threshold value was chosen to strike a balance between accurately classifying runs and minimizing misclassifications. Fig. 5.22 and Fig. 5.23 display the maximum MSE of each run for training and test sets, respectively. With the selected threshold of 0.0025, all test good runs are correctly classified, achieving a 100% accuracy rate. However, there is a misclassification rate of 4% for test bad runs, indicating that one run of bad runs is falsely identified as good run.

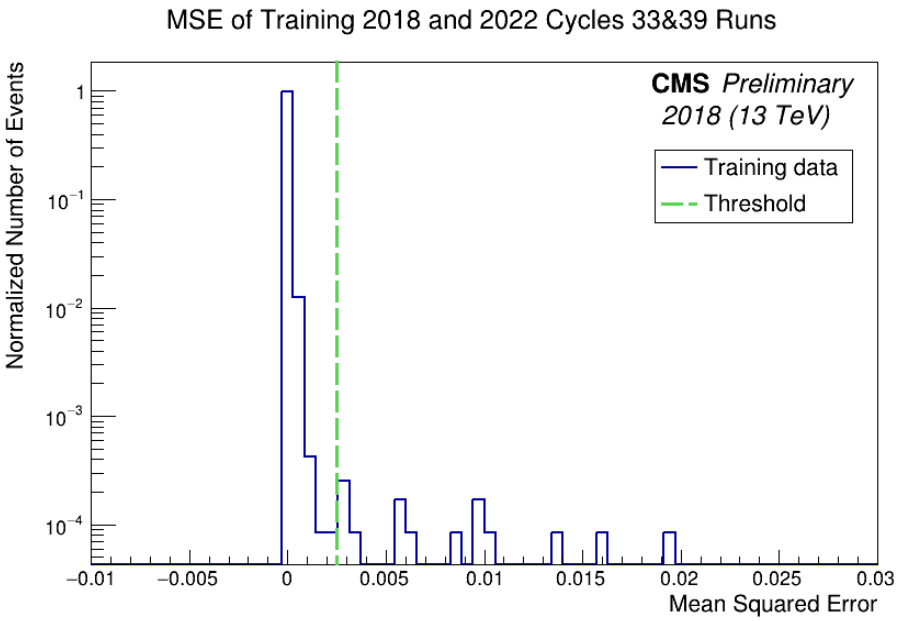


Figure 5.21 The mean squared error of Run-2 2018 and Run-3 2022 Cycles 33 and 39 training good runs for each of the 25 features and the chosen threshold for test runs.

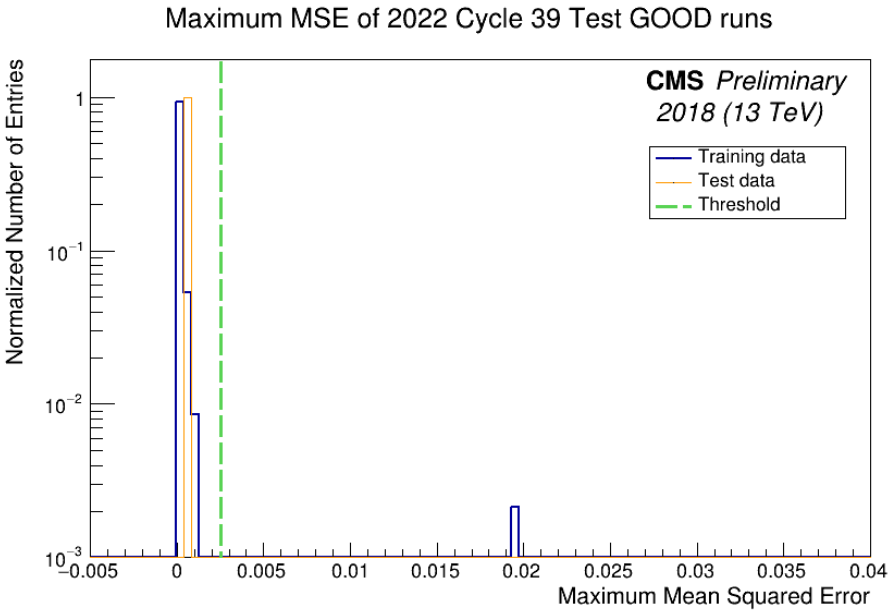


Figure 5.22. Distribution of the max MSE for training good runs in blue and test good runs in orange.

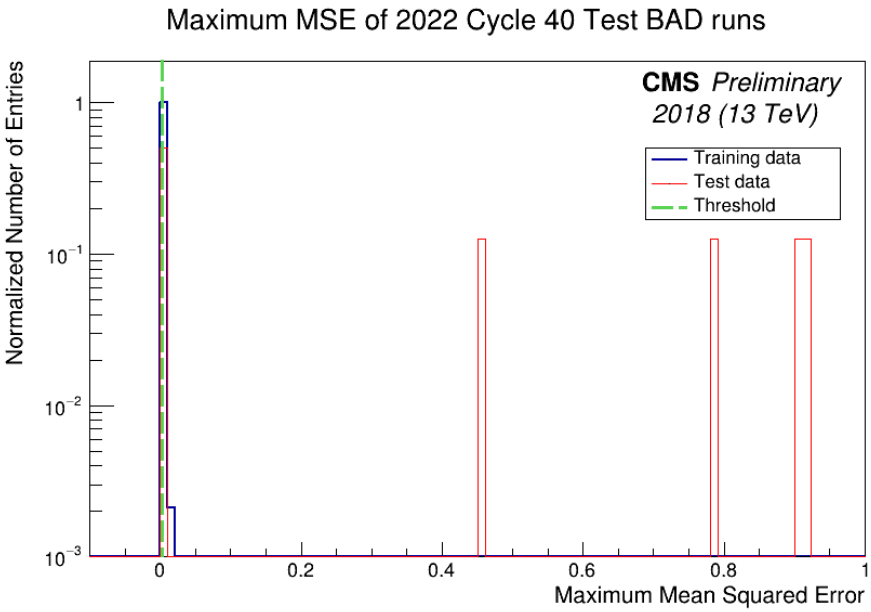


Figure 5.23. Distribution of the max MSE for training good runs in blue and test bad runs in red.

The ROC AUC scores further confirm the effectiveness of the model, with a score of 0.9989 for training runs and 0.9375 for test runs. The corresponding ROC curve is depicted in Fig. 5.24 below.

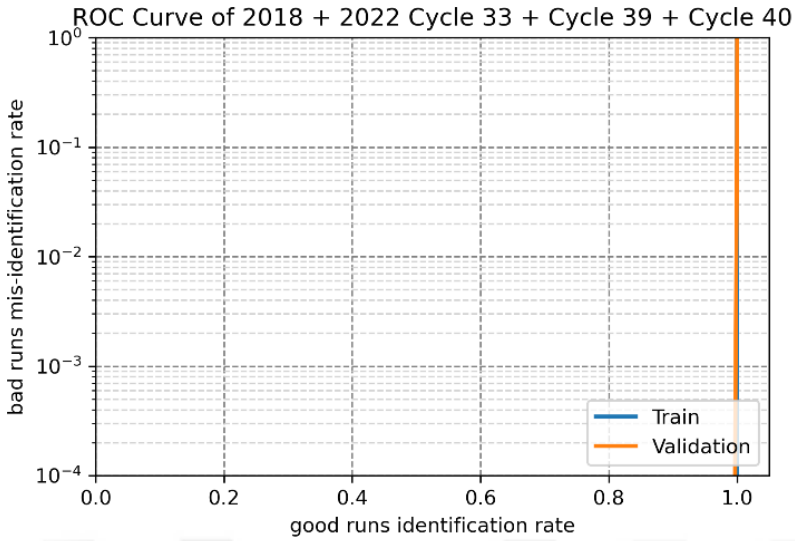


Figure 5.24. ROC curve of binary classification of 2018 and 2022 runs in training and test/ validation sets.

5.2.2. Anomaly Detection with Autoencoders: MET Histogram.

In this study, different types of histograms, including MET histograms and 2-dimensional histograms, will be employed to identify anomalous regions within the detector. To accomplish this, modifications are made to the structure of the AE. Instead of encoding and decoding mean values, the AE will now process entire histograms. So, the input for the study consists of runs with histogram bins. Each bin represents the number of entries or events that fall within a specific range or bin width, depending on the observable being analyzed. This allows the AE to capture the complete distribution information and learn patterns within the histograms.

Starting with MET, we used MET histograms to point out specific runs that suffer from tailing problems. There were 16 runs in total that were marked as bad due to the presence of tails that arise due to detector issues such as LED misfiring [53]. The runs with the identifiers '315784', '315800', '316718', '317339', '317291', '321177', '321295', '321296', '321712', '321730', '321732', '321758', '321760', '321773', '321774', and '321775' exhibited these issues (see Fig. 5.25). To address the problem, particular LSs of each run were excluded based on a common decision made by HCAL and JME after the data taking phase. Initially, the runs were flagged as bad due to the presence of the unremoved LSs. However, a subsequent decision was made to locally rerun the DQM code using the LSs identified as

good by HCAL DPG. As a result, all affected runs were recovered, and their status was changed to good.

Approximately 2.263 fb^{-1} of data were initially affected by the MET tails problems, but through the recovery process, around 2 fb^{-1} of data were successfully regained. One example of the recovery and remarking of one run as good is depicted in Fig. 5.26 providing visual representations of the progress made.

However, in our analysis using the 2018 dataset, we encountered runs with bad LSs. To assess the effectiveness of ML in identifying the tails problems associated with these runs, we opted to train a fully connected autoencoder specifically on this subset. This allowed us to evaluate the ML model's capability in recognizing and flagging the tails issues, providing valuable insights into the power of ML in addressing such problems.

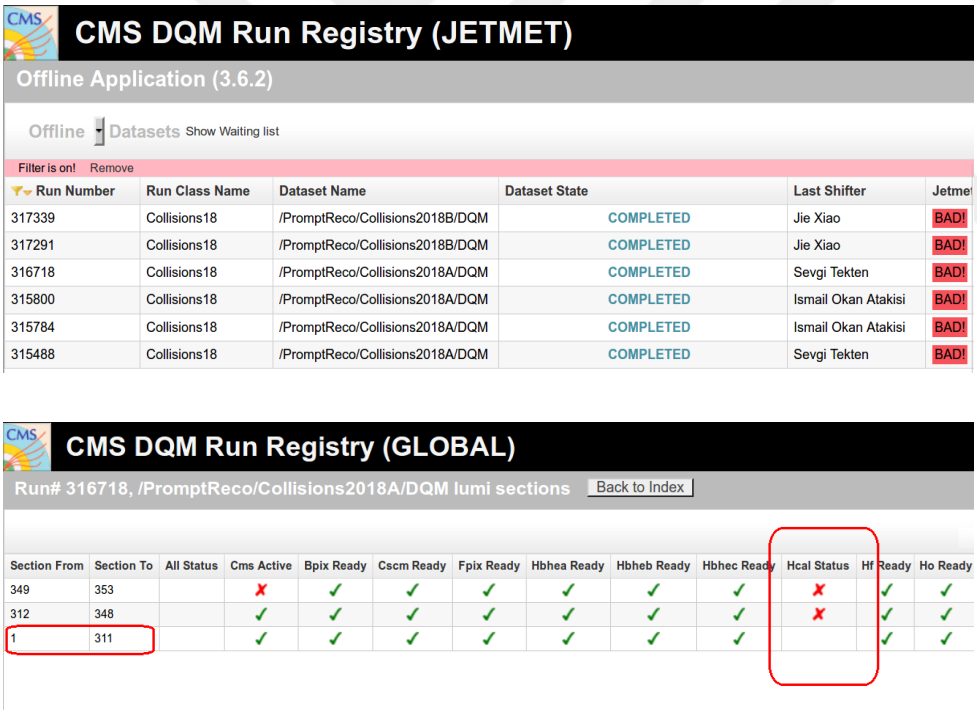


Figure 5.25. A snapshot of the CMS DQM GUI Run registry for runs suffering from MET tails [25].

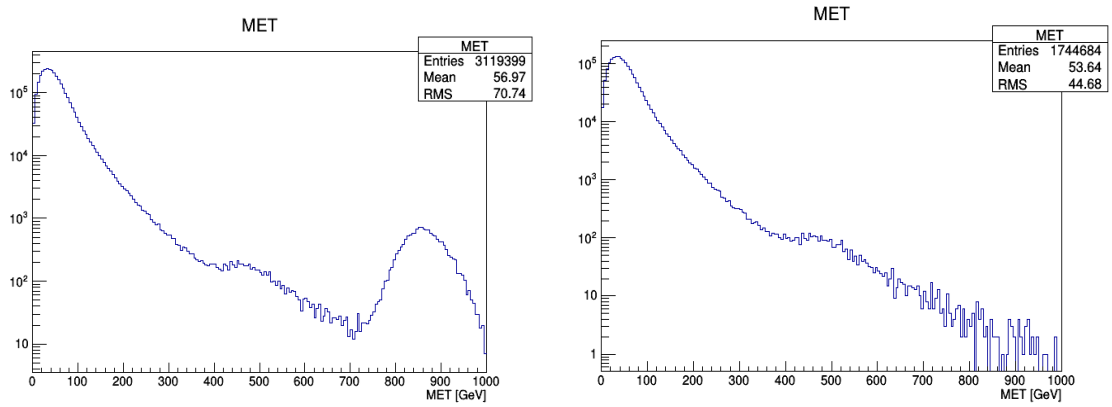


Figure 5.26. MET distributions of run 315488 where the plot on the left has the full LS and the plot on the right has the updated good LS (1-843).

We employed a fully connected AE to train specifically on the 2018 prompt-reco good runs. Then, we tested the trained AE model on bad runs that exhibit MET tails. Each run consists of 200 bins, and to ensure comparability across all runs, we normalized each bin by dividing it by the maximum bin content observed across all runs. We obtained the maximum values from the MET histograms and performed normalization based on this maximum value. Furthermore, we rearranged the values to establish a maximum value, as the absence of normalization would prevent the achievement of an accurate MET distribution.

So, the fully connected autoencoder is trained with 2018 prompt-reco good runs only, then tested with bad runs suffering from MET tails. The loss function used is the mean absolute value for this model. Then, from the maximum of the loss values, we define a threshold to detect the MET tails. As seen in Fig. 5.27 and Fig. 5.28, which present the MET histograms of two runs, one is good and the other is bad, and the loss in logarithmic scale between the input and reconstructed histograms for both runs.

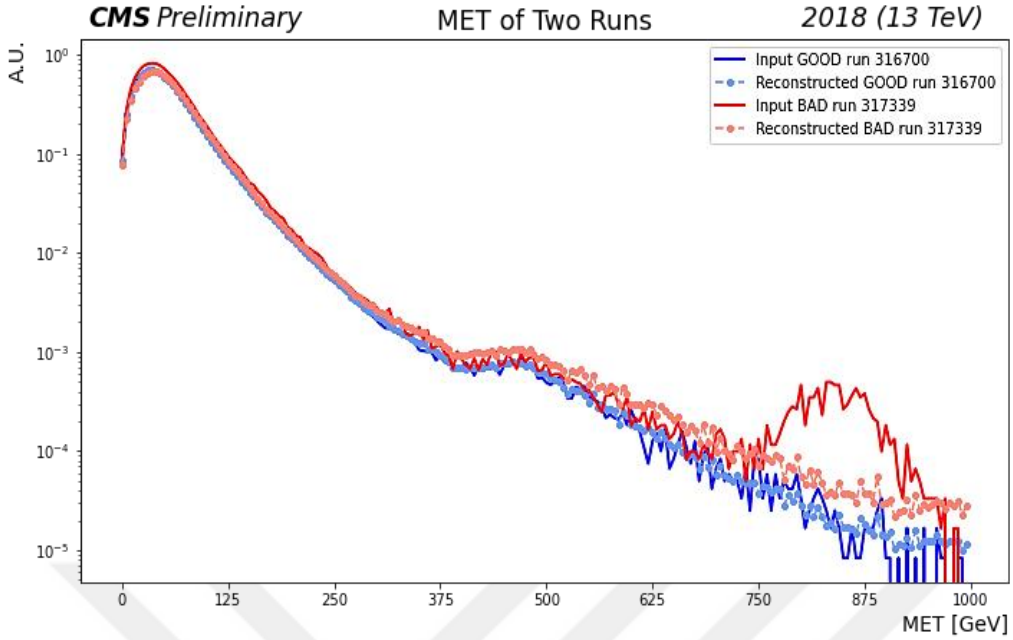


Figure 5.27. MET distributions of two runs: a good run in blue and a bad run in red for both input and reconstructed data.

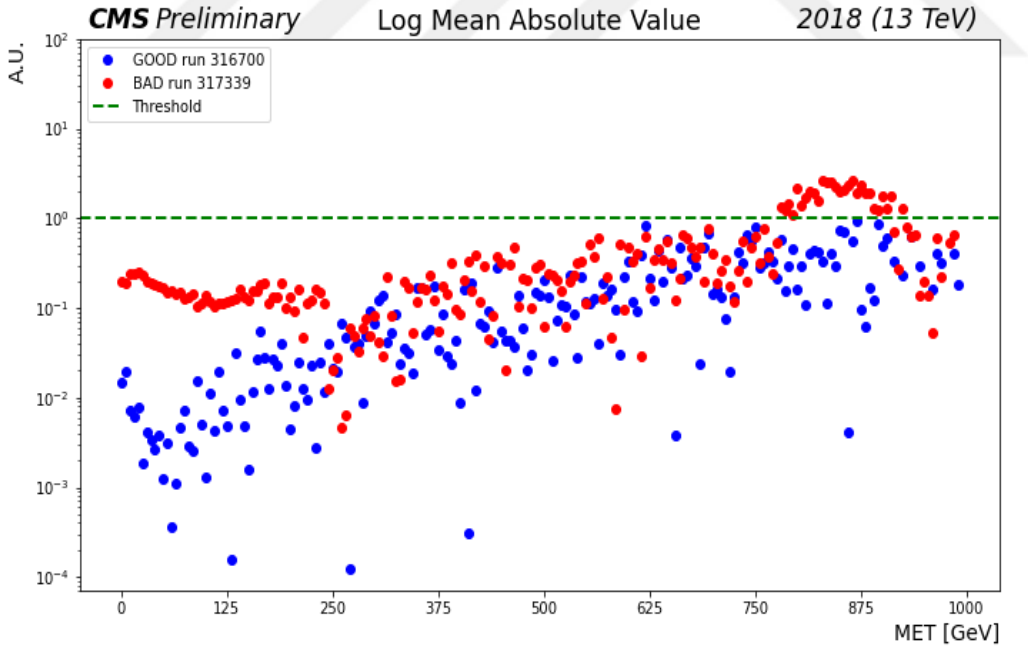


Figure 5.28. The logarithmic scale of the absolute value losses between the input and reconstructed MET distributions of two runs; a good run (blue) and a bad run (red).

As can be seen, in the prompt-reco dataset all runs (good and bad) have a small bump around 400 GeV. This issue is solved in the ultra-legacy dataset, so we also tested with UL dataset, and we got almost same good results as shown below. As seen in Fig. 5.29 and Fig. 5.30

which present the MET histograms of two runs, one is good and the other is bad, and the loss in logarithmic scale between the input and reconstructed histograms for both runs.

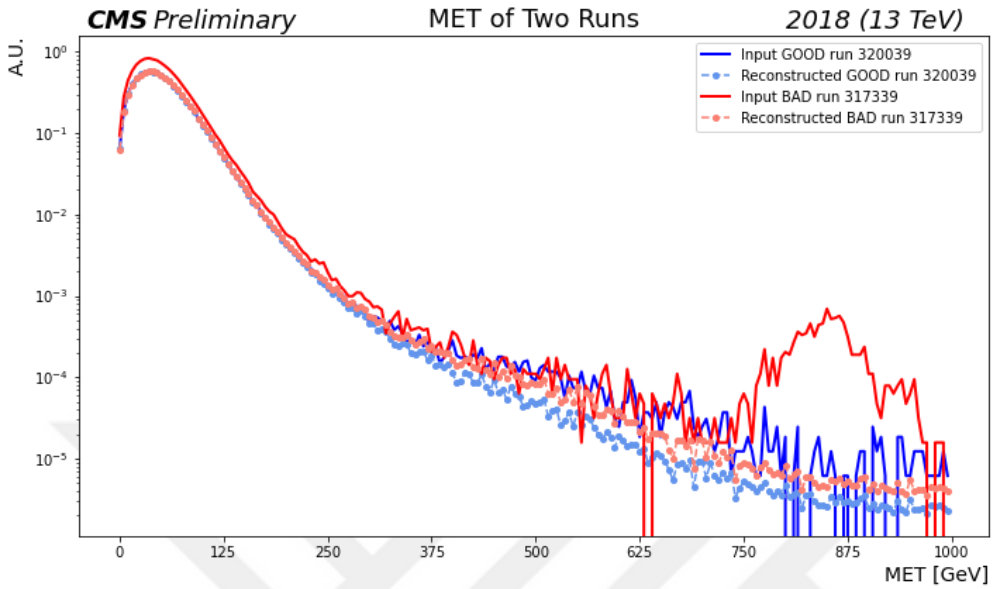


Figure 5.29. MET distributions of two runs: a good run (blue) and a bad run (red) for both input and reconstructed data.

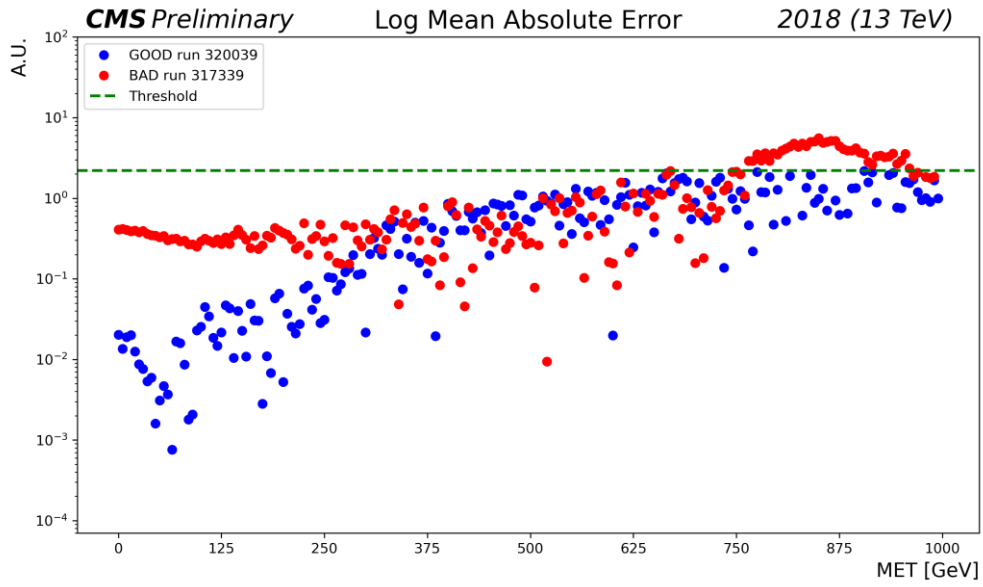


Figure 5.30. The logarithmic scale of the absolute value losses between the input and reconstructed MET distributions of two runs; a good run (blue) and a bad run (red).

5.2.2. Anomaly Detection with Autoencoders: 2-d Histograms

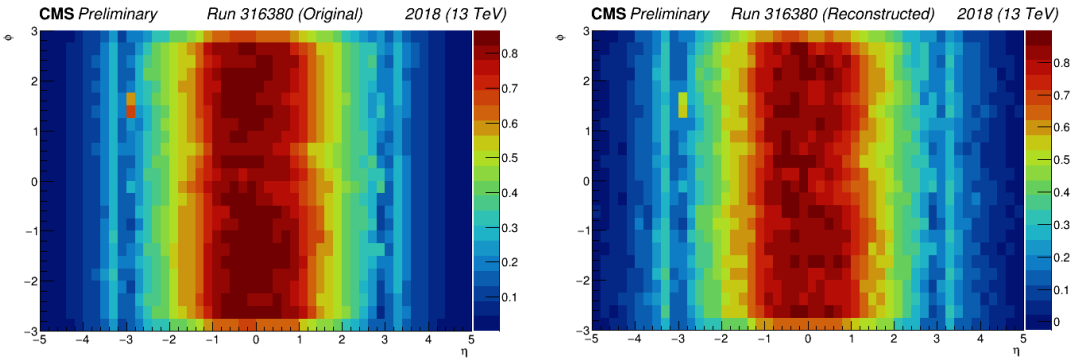
The objective of this study is to leverage an MLP autoencoder to differentiate efficiencies between runs with different Eras and runs within the same Era. Each Era represents a distinct data-taking period characterized by varying detector configurations. By training the model on different histograms, including Phi VS Eta (Run-2 2018), Hadron Occupancy (Run-2 2018), Neutral Hadron EndCap Occupancy (Run-2 2018), and Phi VS Eta (Run-3 2022), we aim to minimize the loss function using mean squared error.

To assess the capabilities of the MLP autoencoder, we conducted tests on two types of datasets. Firstly, we examined runs from the same Era (Era A) to identify potential inefficiencies within that specific Era. Subsequently, we evaluated the model's performance by testing it on runs from different Eras (Era D) to detect inefficiencies and compare efficiencies across Eras.

The results obtained from the MLP autoencoder were highly promising across all four datasets. The model showcased exceptional performance, successfully differentiating efficiencies among different Eras and even within the same Era. This differentiation was achieved through the analysis of the loss map, which provided valuable insights into the variations and discrepancies in efficiencies.

1. Phi VS Eta (Run-2 2018)

First, we train and test with runs from same Era A. The figures below demonstrate the autoencoder result when fed with Era A runs as input and test with runs from same Era A. Figures 5.31 (a), 5.31 (b) and 5.32 show the original image of Phi VS Eta histogram, the reconstructed image of hadron occupancy based on the input image, and the loss map (MSE) of Phi VS Eta respectively. As seen, the loss map shows almost zero loss everywhere, since the autoencoder has reconstructed the input of the same Era very well.



(a) (b)

Figure 5.31. (a) The Original image of PhiVSEta 2018 histogram. (b) The Reconstructed image of PhiVSEta based on the input image.

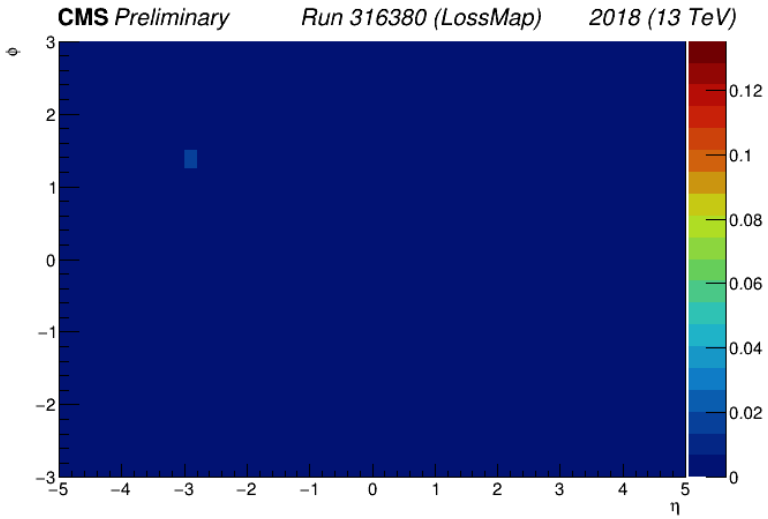
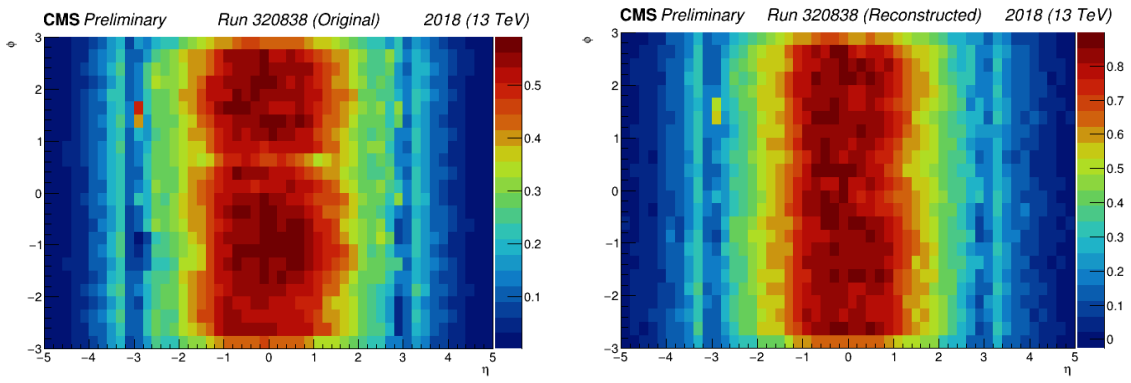


Figure 5.32. The loss map, which represents the mean squared error of PhiVSEta Histogram for one run.

Then, we test with runs from different Era which is Era D. The figures below demonstrate the autoencoder result when fed with Era A runs as input and tested with runs from Era D. Figures 5.33 (a), 5.33 (b) and 5.34 show the original image of Phi VS Eta histogram, the reconstructed image of hadron occupancy based on the input image, and the loss map (MSE) of Phi VS Eta respectively. As seen, the loss map shows a significant loss in the barrel region which indicates a lost sector in that region, so the autoencoder was able to differentiate between different Eras.



(a) (b)

Figure 5.33. (a) The Original image of PhiVSEta 2018 histogram. (b) The Reconstructed image of PhiVSEta based on the input image.

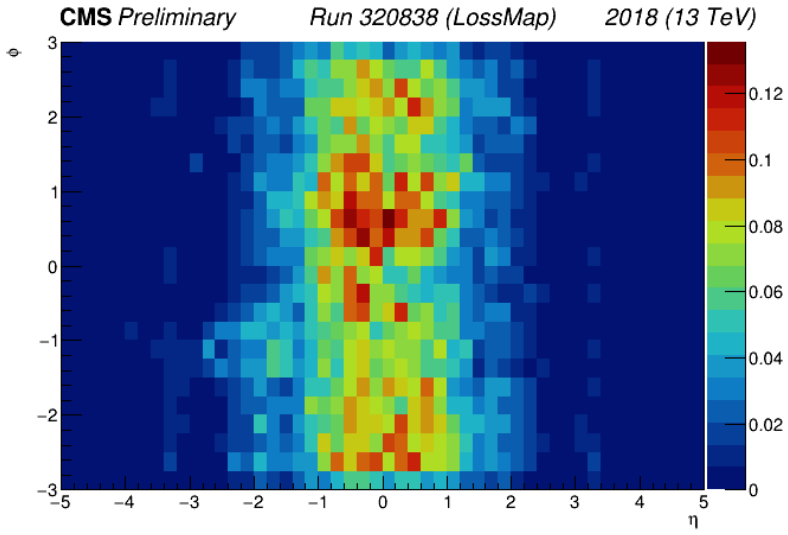
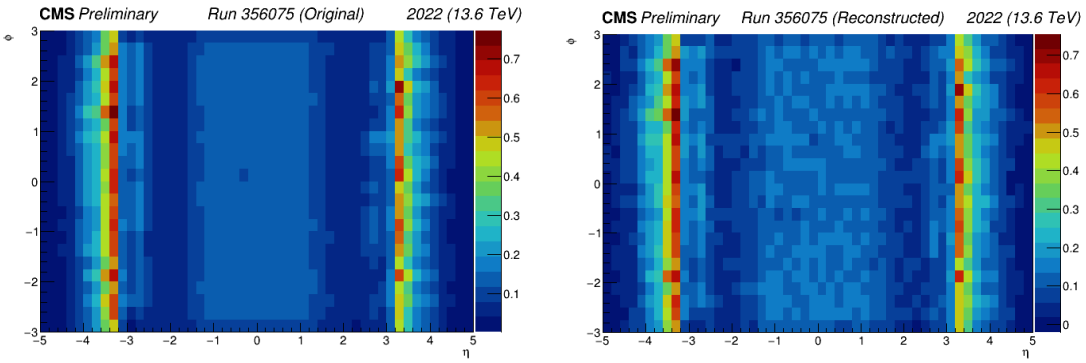


Figure 5.34. The loss map (MSE) of PhiVSEta.

2. Phi VS Eta (Run-3 2022)

First, we train and test with runs from same Cycle which is Cycle 33. The Figures below demonstrate the autoencoder result when fed with Cycle 33 runs as input and tested with runs from same Cycle 33. Figures 5.35 (a), 5.35 (b) and 5.36 show the original image of Phi VS Eta histogram, the reconstructed image of hadron occupancy based on the input image, and the loss map (MSE) of Phi VS Eta. As seen, the loss map shows almost zero loss everywhere, since the autoencoder has reconstructed the input of the same Cycle very well.



(a) (b)

Figure 5.35. (a) The Original image of PhiVSEta 2022 histogram. (b) The Reconstructed image of PhiVSEta based on the input image.

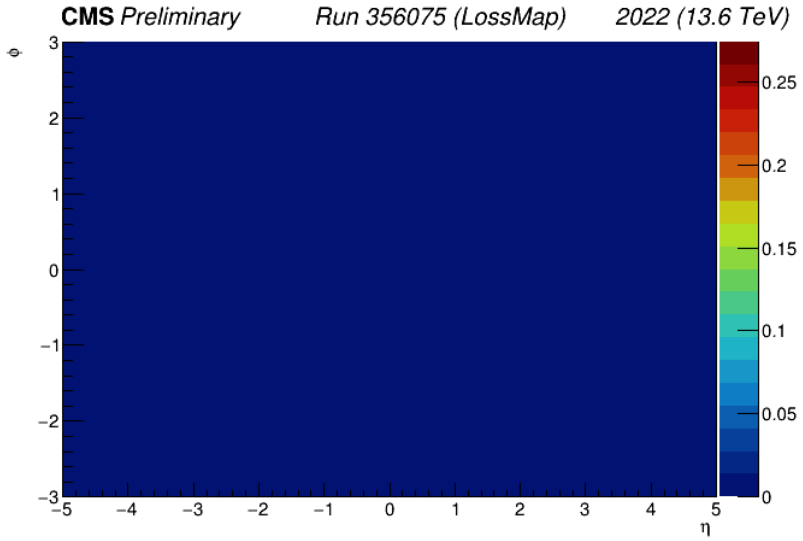
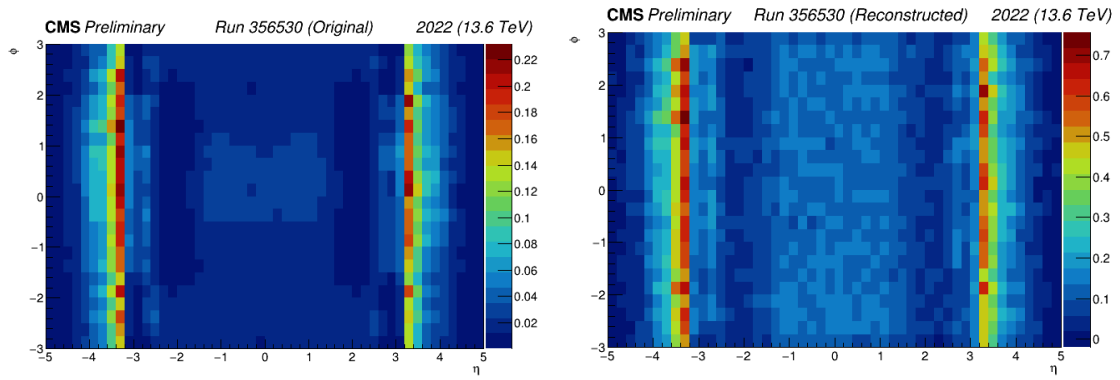


Figure 5.36. The loss map (mean squared error) of PhiVSEta.

Then, we test with runs from different Cycle which is Cycle 34. The Figures below demonstrate the autoencoder result when fed with Era A runs as input and tested with runs from different Era D. Figures 5.37 (a), 5.37 (b) and 5.38 show the original image of Phi VS Eta histogram, the reconstructed image of hadron occupancy based on the input image, and the loss map of Phi VS Eta. As seen, the loss map shows a significant loss in the barrel region which indicates a lost sector in that region, so the autoencoder was able to differentiate between different Cycles.



(a) (b)

Figure 5.37. (a) The Original image of PhiVSEta 2022 histogram. (b) The Reconstructed image of PhiVSEta based on the input image.

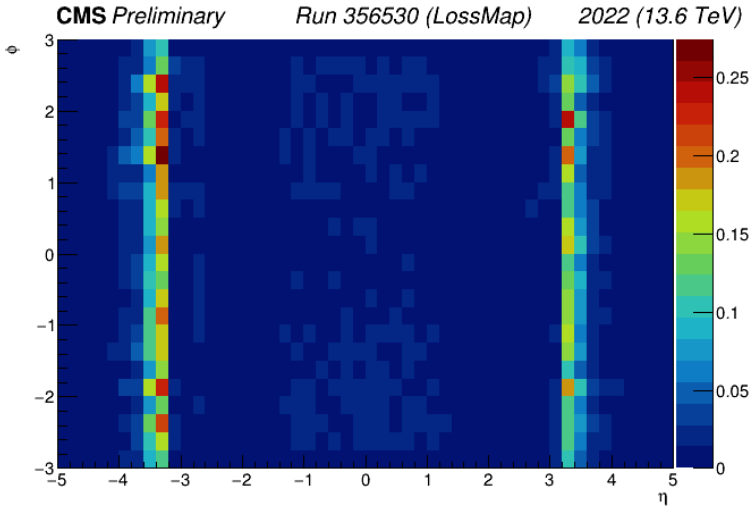
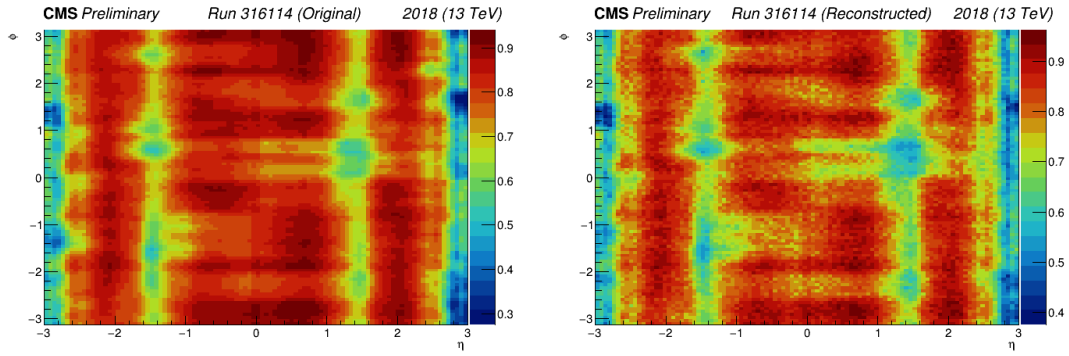


Figure 5.38. The loss map (mean squared error) of PhiVSEta.

3. Hadron Occupancy (Run-2 2018)

First, we train and test with runs from same Era which is Era A. The Figures below demonstrate the autoencoder result when fed with Era A runs as input and tested with runs from same Era A. Figures 5.39 (a), 5.39 (b) and 5.40 show the original image of hadron occupancy histogram, the reconstructed image of hadron occupancy based on the input image, and the loss map (MSE) of hadron occupancy. As seen, the loss map shows almost zero loss everywhere, since the autoencoder has reconstructed the input of the same Era very well.



(a) (b)

Figure 5.39. (a) The Original image of hadron occupancy 2018 histogram. (b) The Reconstructed image of hadron occupancy based on the input image.

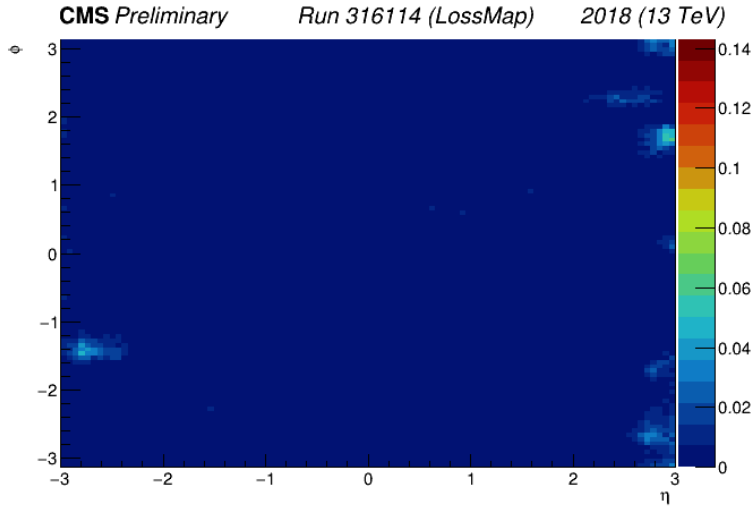
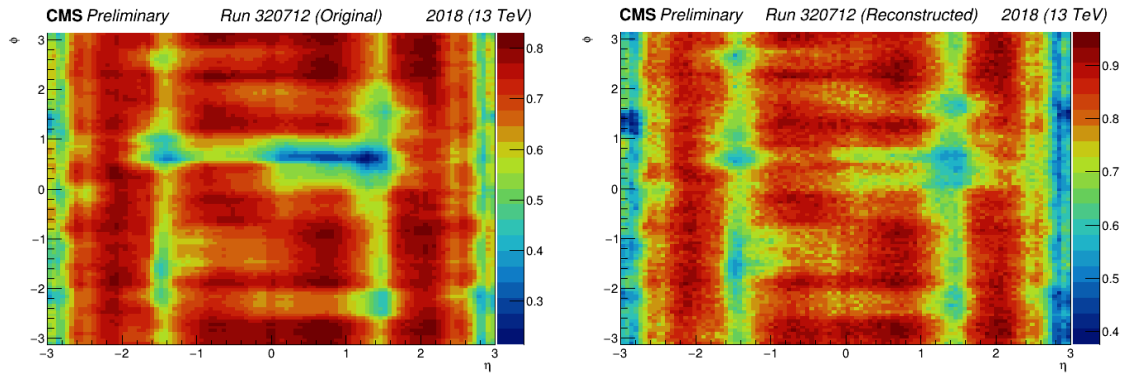


Figure 5.40. The loss map (mean squared error) of hadron occupancy.

Then, we test with runs from different Era which is Era D. The Figures below demonstrate the autoencoder result when fed with Era A runs as input and tested with runs from different Era D. Figures 5.41 (a), 5.41 (b) and 5.42 show the original image of hadron occupancy histogram, the reconstructed image of hadron occupancy based on the input image, and the loss map of hadron occupancy. As seen, the loss map shows a significant loss in the barrel region which indicates a lost sector in that region, so the autoencoder was able to differentiate between different Eras.



(a) (b)

Figure 5.41. (a) The Original image of hadron occupancy 2018 histogram. (b) The Reconstructed image of hadron occupancy based on the input image.

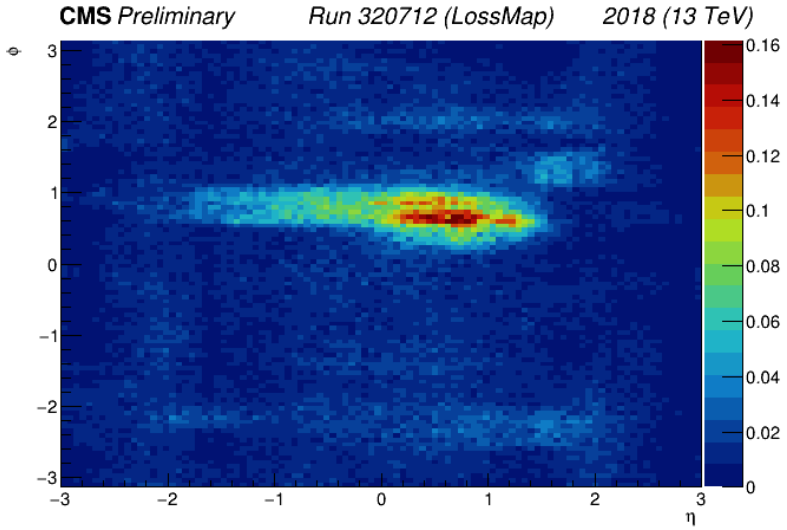
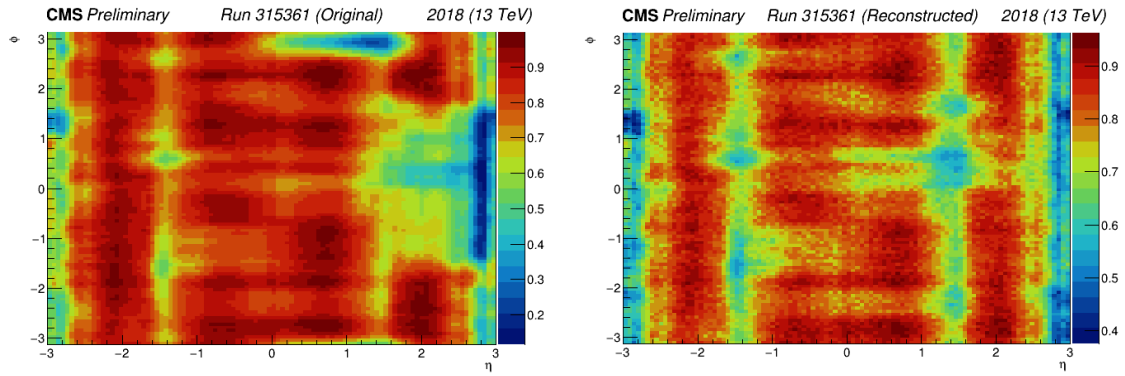


Figure 5.42. The loss map (mean squared error) of hadron occupancy.

Furthermore, we test with problematic runs from same Era A. The Figures below demonstrate the autoencoder result when fed with Era A runs as input and tested with problematic runs from same Era A. Figures 5.43 (a), 5.43 (b) and 5.44 show the original image of hadron occupancy histogram, the reconstructed image of hadron occupancy based on the input image, and the loss map of hadron occupancy. As seen, the loss map shows high loss in some of the detector regions which indicates that this run is problematic, so the autoencoder was able to point out inefficiencies of runs within same Era.



(a) (b)

Figure 5.43. (a) The Original image of hadron occupancy 2018 histogram. (b) The Reconstructed image of hadron occupancy based on the input image.

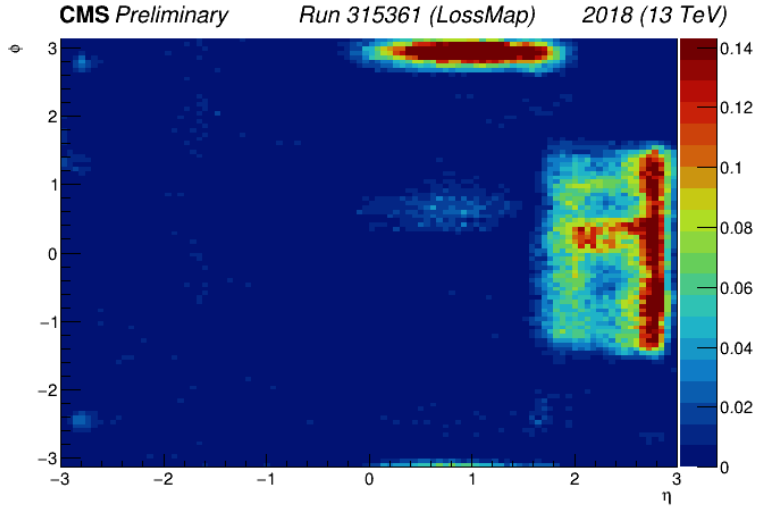
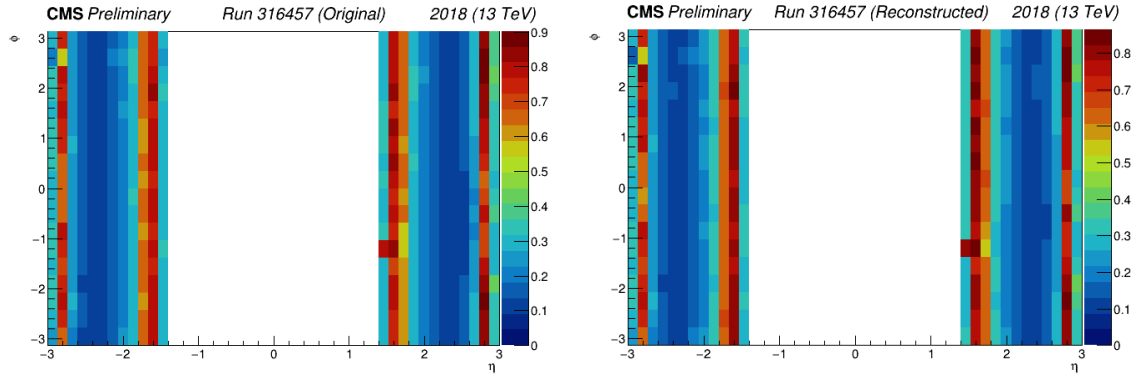


Figure 5.44. The loss map (mean squared error) of hadron occupancy.

4. Neutral Hadron EndCap Occupancy (Run-2 2018)

First, we train and test with runs from same Era A. The Figures below demonstrate the autoencoder result when fed with Era A runs as input and tested with runs from same Era A. Figures 5.45 (a), 5.45 (b) and 5.46 show the original image of Neutral Hadron EndCap Occupancy histogram, the reconstructed image of Neutral Hadron EndCap Occupancy based on the input image, and the loss map (MSE) of hadron occupancy. As seen, the loss map shows almost zero loss everywhere, since the autoencoder has reconstructed the input of the same Era very well.



(a) (b)

Figure 5.45. (a) The Original image of neutral hadron endcap occupancy 2018 histogram. (b) The Reconstructed image of neutral hadron endcap occupancy based on the input image.

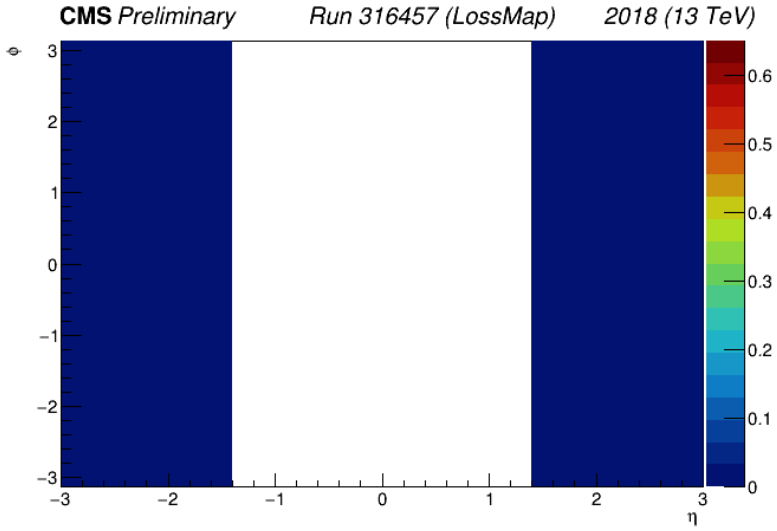
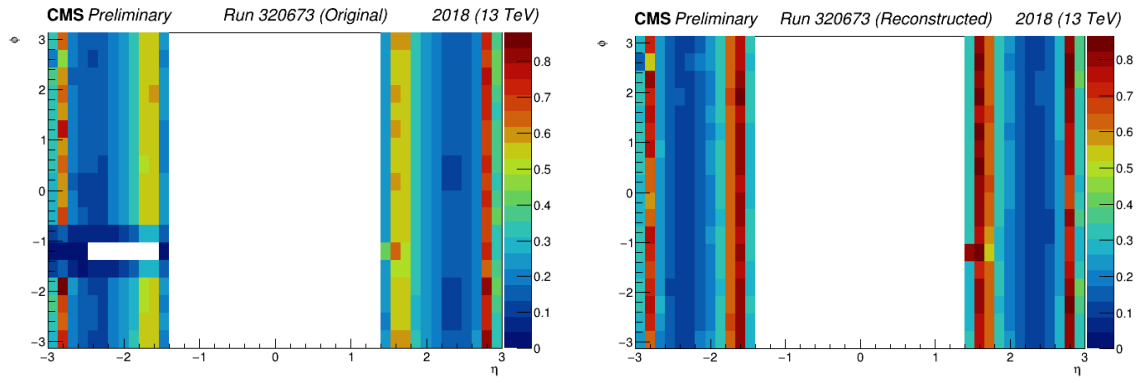


Figure 5.46. The loss map (mean squared error) of neutral hadron endcap occupancy.

Then, we test with runs from Era D. The Figures below demonstrate the autoencoder result when fed with Era A runs as input and tested with runs from different Era D. Figures 5.47 (a), 5.47 (b) and 5.48 show the original image of hadron occupancy histogram, the reconstructed image of Neutral Hadron EndCap Occupancy based on the input image, and the loss map of Neutral Hadron EndCap Occupancy. As seen, the loss map shows a significant loss in the barrel region which indicates a lost sector in that region, so the autoencoder was able to differentiate between different Eras.



(a) (b)

Figure 5.47. (a) The Original image of neutral hadron endcap occupancy 2018 histogram. (b) The Reconstructed image of neutral hadron endcap occupancy based on the input image.

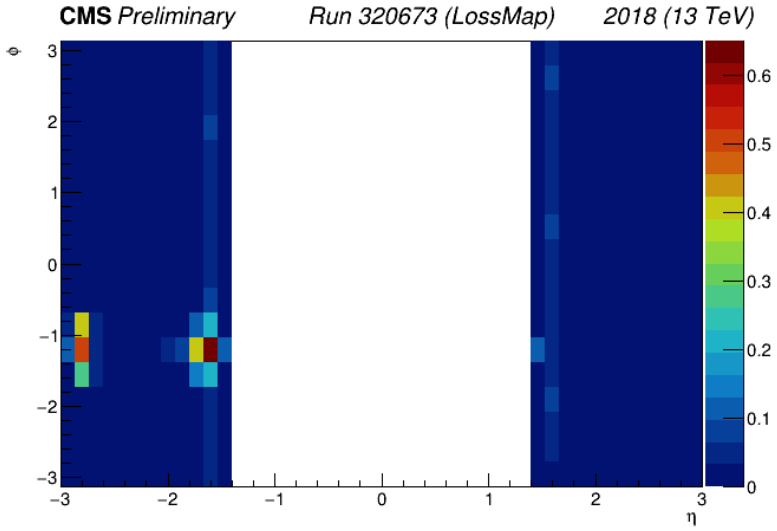


Figure 5.48. The loss map (mean squared error) of neutral hadron endcap occupancy.

Summary and Conclusion

The implementation of data quality monitoring serves as a crucial mechanism to efficiently detect and address any potential issues that may arise with particle physics data, thereby mitigating the risk of data loss in the CMS experiment at the LHC. With this thesis, we showed that with machine learning techniques offline data certification of runs can be done automatically with high efficiency and less time. First, variable reduction with neural network classification was used to detect the most important variables for DC. Then, to determine the effectiveness of the variables in validating runs, several supervised classifiers were trained on the mean values of jet energy fractions, which were selected using the variable reduction approach. Next, in the unsupervised implementation, the fully connected autoencoder was trained on different sets of jet energy fractions where the runs were certified by checking the highest MSE among the reconstructed jet energy fractions. Furthermore, to detect problems in the tails of MET distributions, an MLP autoencoder was trained to reconstruct MET histograms. Finally, an MLP autoencoder was trained on different 2-d histograms to detect inefficiencies arising between runs.

In conclusion, the utilization of unsupervised machine learning techniques enables the certification of runs pertaining to hadronic jets, and the achieved results are comparable to those obtained through supervised machine learning approaches. The flexibility offered by unsupervised machine learning allows for the individual utilization of each JetMET histogram. This approach enables the detection of anomalies by examining discrepancies in the MET distribution, which can reveal incorrect measurements of jet energy within specific detector sectors. Additionally, training on the jet distributions in the Eta and Phi planes allows for the identification of instances where the distributions of jets exhibit variations. Overall, the application of unsupervised machine learning proves valuable in the analysis of hadronic jets and provides insights into the potential presence of anomalies or irregularities.

References

[1] ATLAS Collaboration (2012) Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. Physics Letters B, Volume 716, Issue 1, Pages 1-29.

[2] MissMJ, Cush (2019) Standard Model of Elementary Particles. Wikipedia: PBS NOVA, Fermilab, Office of Science, United States Department of Energy, Particle Data Group.

[3] O. Brüning, H. Burkhardt, S. Myers (2012) The Large Hadron Collider. Progress in Particle and Nuclear Physics, Volume 67, Issue 3, Pages 705-734.

[4] ATLAS Collaboration (2008) The ATLAS Experiment at the CERN Large Hadron Collider. JINST 3 S08003.

[5] CMS Collaboration (2008) The CMS experiment at the CERN LHC. JINST 3 S08004.

[6] LHCb Collaboration (2008) The LHCb Detector at the LHC. JINST 3 S08005.

[7] ALICE Collaboration (2008) The ALICE experiment at the CERN LHC. JINST 3 S08002.

[8] W. Herr, B. Muratori (2006) Concept of luminosity. CERN Accelerator School: Intermediate Accelerator Physics, pp.361-378.

[9] N. Ellis (2010) Trigger and data acquisition. arXiv:1010.2942v.

[10] CMS Collaboration (2000) CMS TriDAS project: Technical Design Report, Volume 1: The Trigger Systems. Technical Design Report CMS.

[11] CMS Collaboration (2002) CMS The TriDAS Project: Technical Design Report, CMS Collaboration Volume 2: Data Acquisition and High-Level Trigger. CMS trigger and data-acquisition project. Technical Design Report CMS.

[12] CMS Collaboration, CMS detector design. https:///cms.cern/detector [Online; accessed May 06, 2023].

[13] N. Izaak (2021) [Online; accessed May 07, 2023] CMS coordinate system. https://github.com/IzaakWN/CodeSnippets/tree/master/LaTeX/TikZ.

[14] F. Beaudette (2013) The CMS Particle Flow Algorithm. arXiv:1401.8155.

[15] D. Barney (2016) CMS Detector Slice [Online; accessed May 06, 2023]. http://cds.cern.ch/record/2120661/?ln=de.

[16] CMS Collaboration (2011) Determination of Jet Energy Calibration and Transverse Momentum Resolution in CMS. arXiv:1107.4277v1.

[17] M.Cacciari, G. P. Salam, G. Soyez (2008) The anti-kt jet clustering algorithm. arXiv:0802.1189.

[18] CMS Collaboration (2011) Missing transverse energy performance of the CMS detector, arXiv:1106.5048.

[19] CMS Collaboration (2009) Particle–Flow Event Reconstruction in CMS and Performance for Jets, MET and Taus. CMS Physics Analysis Summary: CMS-PAS-PFT-09-001.

[20] CMS Collaboration (2007) Performance of Jet Algorithms in CMS. CMS Physics Analysis Summary: CMS-PAS-JME-07-003.

[21] CMS Collaboration (2009) Performance of Track-Corrected Missing MET in CMS. CMS Physics Analysis Summary: CMS-PAS-JME-09-010.

[22] F. De Guio, CMS Collaboration (2014) The CMS data quality monitoring software: experience and future prospects. J. Phys.: Conf. Ser. 513 032024.

[23] V. Azzolini, B. Van Besien, D. Bugelskis, T. Hreus, K. Maeshima, J. M. Fernandez A. Norkus, J. Patrick, M. Rovere, A. Schneider (2019) The Data Quality Monitoring software for the CMS experiment at the LHC: past, present and future. EPJ Web Conf. 214 02003.

[24] R.Brun, F.Rademakers, S.Panacek (2000) ROOT, an object oriented data analysis framework. CERN School of Computing: pp.11-41.

[25] V. Rapsevicius (2011) CMS Run Registry: Data certification bookkeeping and publication. System Journal of Physics: Conference Series vol 331 (IOP Publishing) p 042038.

[26] Anton Karneyeu (2018) Data Quality Monitoring (DQM) for physics analysis. [Online; accessed May 11, 2023].
https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookDQM.

[27] L. Tuura, G. Eulisse, A. Meyer (2010) CMS data quality monitoring web service. Journal of Physics: Conference Series vol 219 (IOP Publishing) p 072055.

[28] CMS Collaboration (2020) Dependence of inclusive jet production on the anti-kT distance parameter in pp collisions at $\sqrt{s} = 13\ TeV$. Published for SISSA by Springer.

[29] CMS Collaboration (2023) An AutoEncoder-based Anomaly Detection tool with a per-LS granularity. CMS DP -2023/010.

[30] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D.,

Brucher, M., Perrot, M., & Duchesnay, E. (2011) Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. 12, 2825-2830.

[31] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., & Zheng, X. (2016) TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. arXiv:1603.04467.

[32] F. Chollet, (2015) Keras. GitHub repository. https://github.com/fchollet/keras.

[33] T. Cover, P. Hart (1967) Nearest neighbor pattern classification. IEEE Transactions on Information Theory. 13(1), 21-27.

[34] Vikramkumar, Vijaykumar B, Trilochan (2014) Bayes and Naive Bayes Classifier. arXiv:1404.0933.

[35] C. Cortes, V. Vapnik (1995) Support-vector networks. 20(3), 273-297.

[36] N. Avinash (2019) Support Vector Machines with Scikit-learn Tutorial. [Online; accessed May 14, 2023]. https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python.

[37] C. Chang, C. Lin (2011). A library for support vector machines. ACM Transactions on Intelligent Systems and Technology (TIST), 2(3), 27.

[38] Y. LeCun, Y. Bengio, and G. Hinton (2015) Deep learning. Nature, vol. 521, no. 7553.

[39] C. Burnett (2006) An example artificial neural network with a hidden layer. Wikimedia Commons, 753183698.

[40] C.E Nwankpa, I. Winifred, G. Anthony, M. Stephen (2018) Activation Functions: Comparison of Trends in Practice and Research for Deep Learning. arXiv:1811.03378v1.

[41] G. Klambauer, T. Unterthiner, A. Mayr, S. Hochreiter (2017) Self-normalizing neural networks. Advances in neural information processing systems (pp. 971-980).

[42] S. Ioffe, C. Szegedy (2015) Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv:1502.03167.

[43] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov (2014) Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research, vol. 15, pp.

[44] D.P. Kingma, J. Ba (2015) Adam: A method for stochastic optimization. In Proceedings of the 3rd International Conference on Learning Representations (ICLR).

[45] J. Duchi, E. Hazan, Y. Singer (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 2121-2159.

[46] T. Tieleman, G. Hinton (2012) Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning, 4(2), 26-31.

[47] PT. de Boer, D.P. Kroese, S. Mannor (2005) A Tutorial on the Cross-Entropy Method. Ann Oper Res 134, 19–67.

[48] G. Cybenko, D.P. O'Leary, J. Rissanen (1999) The Mathematics of Information Coding, Extraction and Distribution. Springer 9780387986654.

[49] K. Murphy (2012) Machine Learning: A Probabilistic Perspective. MIT, ISBN 978-0262018029.

[50] D.E. Rumelhart, G.E Hinton, R.J Williams (1986) Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chap. Learning Internal Representations by Error Propagation. MIT Press, pp. 318–362.

[51] N. Subhashini (2020) Multilayer Perceptron with Auto encoder enabled Deep Learning model for Recommender Systems. Future Computing and Informatics Journal: Vol. 5: Iss. 2, Article 3.

[52] K. Joscha (2023) Public CMS Luminosity Information [Online; accessed May 15, 2023]. https://twiki.cern.ch/twiki/bin/view/CMSPublic/LumiPublicResults#2018_proton_proton_collisions.

[53] CMS Collaboration (2018) Overview of the CMS detector performance at LHC Run 2. CMS-CR-2018.

JAWAHER

ALTORK

Objective

I am a recent graduate majoring in physics. I have shown myself to be self-motivated, committed and determined in achieving my goals. I have strong analytical and computational skills. I possess excellent communication skills and can relate to a wide range of people. I am willing to learn and develop new scientific skills, as in the future I want to be a researcher in Physics.

Education

Bachelor Degree in Physics | University of Jordan Amman, Jordan

09/2016 – 07/2020

GPA: 3.53/4.0

Relevant coursework: Quantum Mechanics, Classical Mechanics, Electromagnetism, Nuclear Physics, Radiation Physics, Elementary Particles, Mathematical Physics.

Honors: Top 10%

Key achievements:

- Designed and built an Air Conditioner.
- Gave a presentation in the field of Particle Physics, particularly about Jet Physics.
- Wrote a review paper about Quantum Information and Quantum Computation.

Master Degree in Physics | Marmara University Istanbul, Turkey

11/2020 – 07/2023

GPA: 3.79/4.0

Thesis Title: Machine Learning techniques for JetMET data certification of the CMS detector.

Supervisor: Prof. Dr. Mithat Kaya.

Relevant coursework: Quantum Mechanics, Classical Mechanics, Electromagnetic Theory, High Energy Physics, Physics of Semiconductor Devices, Statistical Mechanics.

Key achievements:

- Presented my thesis work in: LHCP conference in Belgrade, Serbia 2023, CMS JetMET workshop in Florence 2022, CMS JetMET workshop in Brussels 2023, and ITU 2nd international graduate research symposium in Istanbul, Turkey 2023.
- Published my thesis work as a CMS Detector Performance (DP) note.
- First-Class Honors.

Experience

Research Intern | Deutsches Elektronen-Synchrotron DESY Hamburg, Germany

JULY 2022 – SEPTEMBER 2022

- Participated in a series of lectures related to the research done at DESY.
- Worked on a research project about “Reconstruction of Hadronically Decaying Top Quarks with Unsupervised Self-Organizing Maps” Supervised by: Freya Blekman, Matthias Komm, and Gabriele Milella.

Volunteer | King Hussein Cancer Center Amman, Jordan

OCTOBER 2017 – NOVEMBER 2019

My role included teaching kids to draw, playing with them, assisting in special events and trying to help make the hospital experience a little less stressful for them.

Volunteer | مبادرة ترميم الانسانية (Tarmeem) Amman, Jordan

MAY 2018 – JANUARY 2020

I had the chance to meet wonderful people and learn from their experience. Also, working with the team has developed my social skills. Our role in this group was giving moral support to elderly living in care homes and organizing activities to help people in need.

Volunteer | Istanbul&I Istanbul, Turkey

April 2022 – April 2023

I participated in different activities with Istanbul&I community such as: cleaning animal shelters, cooking for homeless people, organizing and planning social events. Won volunteer of the month of June 2022.

Skills

- Programming languages: C++, ROOT, Python.
- Experience in data analysis libraries: Numpy, Pandas, Matplotlib.
- Knowledge of Artificial Intelligence and Machine Learning libraries: Scikit-Learn, Tensorflow, Keras.
- Microsoft Office: Word, OneDrive, PowerPoint.
- Spreadsheets: Excel and Google Sheets.
- Languages: Arabic (Native), English (Competent), Turkish (intermediate), Korean (Basic), Japanese(Basic).
- Outstanding written and oral communication.
- Ability to work collaboratively in a team environment.
- Time management.
- Passion for learning, solving problems, and challenging the status quo.
- Strong scientific writing skills.
- Highly motivated.

Hobbies & Interests

- Horseback riding.
- Traveling and exploring other cultures.
- Learning languages.
- Playing guitar.
- Drawing and painting.
- Cooking and baking.