Università degli studi di Salerno

Dipartimento di Fisica "E.R. Caianiello"

Dottorato di Ricerca in Matematica, Fisica e applicazioni

XXX ciclo, 2014/2017

Tesi di dottorato in

Event triggering and deep learning for particle identification in KM3NeT

Dottorando:

Chiara De Sio

Tutor:

Prof. Cristiano Bozza

Coordinatore:

Prof. Roberto Scarpa

"*Do. Or do not. There is no try*"

Master Yoda

Abstract

Chiara De Sio

Event triggering and deep learning for particle identification in KM3NeT

Neutrino astronomy experiments like KM3NeT allow to survey the Universe leveraging the properties of neutrinos of being electrically neutral and weakly interacting particles, making them a suitable messenger. Observing neutrino emission in association with electromagnetic radiation allows evaluating models for the acceleration of particles occurring in high energy sources such as Supernovae or Active Galactic Nuclei. This is the main goal of the ARCA project in KM3NeT. In addition, KM3NeT has a program for lower energy neutrinos called ORCA, aimed at distinguishing between the scenarios of "normal hierarchy" and "inverted hierarchy" for neutrino mass eigenstates.

The KM3NeT Collaboration is currently building a network of three Cherenkov telescopes in the Mediterranean sea, in deep water off the coasts of Capopassero, Italy; Toulon, France, and Pylos, Greece. The water overburden shields the detectors from down-going charged particles produced by the interactions of cosmic rays in the atmosphere, while up-going neutrinos that cross the Earth are the target of the observation. Cosmic rays are a background to the KM3NeT signal, usually discarded by directional information. Nevertheless, they provide a reliable reference to calibrate the detector and work out its effective operating parameters, namely direction and energy of the incoming particles.

Estimation of tracking capabilities is directly connected to the evaluation of the ability of the experiment to detect astrophysical point-like sources, i.e. its discovery potential. Being able to distinguish among the three neutrino flavours, or between neutrinos and muons, as well as estimating the neutrino direction and energy, are the main goals of such experiments. Trigger and reconstruction algorithms are designed to separate the signal from background and to provide an estimation for the above mentioned quantities, respectively.

This work describes an innovative approach, based on the application of Deep Learning models, to perform event classification and interaction parameters estimation. KM3NeT simulated events are used as input data to train Neural Network models, capable of classifying the neutrino events based on their shape and the time distribution of the signal hits produced in the detector. In particular, triggered events are fed into Convolutional Neural Network models specifically designed to accomplish 4 different tasks, namely: "up-going" and "down-going" neutrinos classification; ν_μCC and ν_eCC interactions classification; energy and direction estimations of simulated neutrinos. The Convolutional Neural Network models have been implemented using the Keras deep learning framework, a high-level neural networks API designed for easy and fast prototyping which runs seamlessly on CPU and GPU[1].

Since the selection criteria for the events affect the quality of the reconstruction, a preliminary study on the trigger conditions has been conducted to ensure the purity of the selected events.

The developed Deep Learning models have been tested on a dataset consisting of $258,879$ ν_μCC and ν_eCC simulated events, achieving an accuracy of 93.3% for the up-going/down-going classification and 92.8% for the ν_μCC/ν_eCC classification. For the estimation of the neutrino energy and the cosine of the zenith angle of the neutrino direction, the obtained mean squared errors are 0.22 and 0.03, respectively.

The performance of the Neural Network models for energy and direction estimations, as well as up-going/down-going classification have been compared to those calculated by the official reconstruction algorithm currently in use in the KM3NeT reconstruction software pipeline. In this case, only ν_μCC events have been used since the considered reconstruction algorithm, namely `JGandalf`, is based on the assumption of a track-like event shape. The results obtained with the two different approaches are comparable. Nevertheless, the capability of the Neural Networks of performing these estimations directly from raw data constitutes a promising approach in terms of computational times and resources required.

[1]Graphical Processing Unit.

Acknowledgements

First of all, I would like to thank my tutor Prof. Cristiano Bozza for the inputs and the suggestions, and for always setting me on the right track and letting me follow my path. I would also like to express my gratitude to the other members of the “Giorgio Romano” Astro-particle physics and Nuclear Emulsion Group of the University of Salerno, in particular Prof. Giuseppe Grella for admitting me in the group a few years ago, and for his constant encouragement and support. I would also like to thank Simona Maria Stellacci and Maurizio Di Marino for their presence and friendship and for all the moments shared in the laboratory.

I want to thank the KM3NeT Collaboration members for their direct and indirect contributions to this work: every single aspect of the experiment they've been working on made this thesis possible; all their questions, comments and suggestions during the meetings helped me improve this work. In particular, I would like to thank Prof. Christos Markou and Prof. Ekaterini Tzamarioudaki for their precious comments in reviewing the first draft of this manuscript. Their suggestions and point of view helped me to better refine my work.

A special thank goes to the Erlangen group, in particular Prof. Uli Katz and Dr. Kay Graf, who welcomed me in their Department and supervised my work during my visiting period at ECAP. I really enjoyed the six months spent there. Feeling part of that group made me enjoy this Ph.D. even more, and the weekly meetings were always a source of inspiration.

I want to thank my family for their support in every situation, and my friends, who always believe in me and never let me feel alone. I've learnt that distance and time do not really count, as long as friendship is real.

Finally, I would like to thank V., who made everything possible in the last three years, including this.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Overview

KM3NeT events are suitable to be analysed with Deep Learning techniques, which are capable of extracting global information from raw data. In this work, Neural Network techniques are used to classify neutrino interactions and to estimate event parameters according to event features that are automatically defined and extracted by a Neural Network. A brief introduction of the neutrino astronomy and of the KM3NeT experiment will be presented in Chapter 1. A preliminary study on the trigger conditions has been conducted to optimise the event detection and will be described in Chapter 2. The Machine Learning and Deep Learning techniques will be presented in Chapter 3, along with the details of the applications chosen for the purpose of KM3NeT. Finally, in Chapter 4 the models constructed and the results will be presented and discussed, and compared to the performance of the official reconstruction algorithms.

1.1 Neutrino astronomy

In the last decades, neutrino astronomy has been developed to complement the traditional branches of astronomy. Commonly, photons are the carriers of information in astrophysics. Such particles have the advantages of being electrically neutral, easy to detect, abundant in the universe and have a wide spectrum of energy. Furthermore, they carry information about the chemical composition of the observed structures. On the other hand, photons cannot offer any direct insight into some astrophysical high energy sources, such as the dense and hot matter in the centre of the stars and other astrophysical energy sources, which are completely opaque to them. Moreover, high energy photons are partially absorbed by interstellar dust, and interact with infrared radiation background and the cosmic microwave background radiation (CMB) to create electron-positron pairs. Therefore, a different messenger capable of carrying information at greater distances, is needed for the purpose of studying such emitters, and is the main ingredient in a new branch of astronomy: neutrino astronomy. Being electrically neutral, stable and weakly interacting, the neutrino travels long distances undeflected by magnetic fields, carrying information about distant high energy sources, and penetrating regions generally opaque to photons. Over the last 40 years, several astrophysical sources have been studied because they are known neutrino emitters, such as the Sun [1]. On the other hand, astrophysical sources of high-energy neutrinos have not been directly observed yet, whereas several models exist that predict the neutrino emission. The main idea is that information on the features of such sources can be inferred from the properties of cosmic rays, namely protons and nuclei accelerated by astrophysical objects and travelling through cosmic distances [2]. Neutrino astronomy opens a new window to the Universe, by investigating the nature of large and powerful particle accelerators, with the goal of finding an answer to the questions about the origin, the propagation and the acceleration mechanisms of cosmic rays. The exact composition of cosmic rays is still debated, due to the difficulty of an accurate measurement of the mass of each particle impinging the Earth atmosphere. Most estimations, though, agree on the composition of the most prevalent particles: 85% protons, 12% helium, 1% of heavier nuclei ($Z>3$). The remaining part is composed of other particles (e.g. electrons), and a small fraction is represented by anti-particles, such as anti-protons and positrons [3]. The energy spectrum is a power law which extends over 12 orders of magnitude, from around 1 GeV up to extremely high energy values, exceeding 10^{20} eV, as shown in Figure 1.1. The lower limit of such spectrum is set by the interaction of cosmic rays (CRs) with fluctuations of the solar wind magnetic fields, rejecting charged particles coming from outside the solar system, hence causing a reduction of the flux reaching the Earth by a factor 10^8 [4]. Above a few GeV, the spectrum follows a power law $dN/dE \approx E^{-2.7}$, which holds to a good approximation up until the so-called *knee*, around 10^{15} eV. Above the knee, for 3 orders of magnitude in energy the flux falls more steeply, following a power law of E^{-3}, up to the so-called *ankle*, where it becomes less steep again (back to $E^{-2.7}$). The changes in the spectral index (i.e. the exponent of the power law, generally indicated with γ) are due to differences, with the increasing energy, in the origin, in the propagation or in the composition of the cosmic rays. Above 10^{20} eV, the composition and origin of cosmic rays is less known, due to low statistics and consequently high experimental

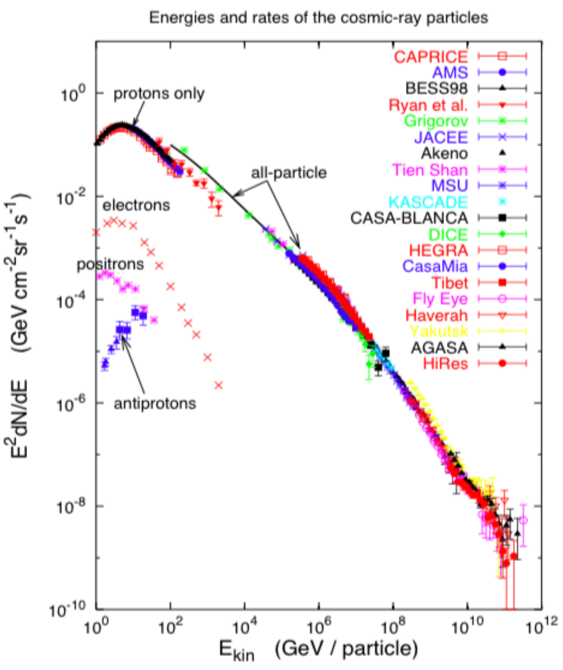


FIGURE 1.1 Cosmic rays all-particle spectrum as measured by different experiments. Shown in [4].

uncertainties. However, the knowledge of such very energetic propagating particles, also known as Ultra High Energy Cosmic Rays (UHECR), represents an important source of information about the possibility of surveying the space using ultra-high energy protons as carriers. In fact, protons with energies greater than 10^{18} eV are most likely of extra-galactic origin, and their energy and charge lets them propagate approximately undeflected, pointing back to their origin. Heavier nuclei, on the other hand, would be deflected by the Galactic magnetic field $B_G \approx 3\ \mu G$, causing particles to describe helical trajectories with a Larmor Radius $r_L = E/(ZeB_G)$. However, at such high energies another effect has to be taken into account when considering the propagation of particles in the Universe: above 10^{19} eV, protons interact with the Cosmic Microwave Background radiation (CMB), consequently losing energy via the resonant pion production expressed by:

$$p + \gamma \to \Delta^+ \to \pi^0 + p, \tag{1.1}$$

$$p + \gamma \to \Delta^+ \to \pi^+ + n. \tag{1.2}$$

A rough calculation, considering the energy of the CMB of 2.7 K, the threshold energy for the p/γ interaction ($E_p \approx 6\cdot 10^{19}$ eV), and the cross section $\sigma_{p/\gamma} \approx 100\ \mu$barn, leads to an absorption length for the UHE protons in the Universe of less than 100 Mpc, i.e. shorter than the distance between cosmological sources and the Earth. This effect is known as the Greisen-Zatsepin-Kuz'min (GZK) limit [5].

While primary cosmic ray nuclei do not travel in straight lines, secondary photons and neutrinos point back to their sources, allowing their identification as cosmic accelerators. As reported above, accelerated protons interact in the surroundings of the Cosmic Ray emitter with photons predominantly via the Δ^+ resonance, producing pions in the final state. Moreover, protons will interact also with the surrounding matter (protons, neutrons and nuclei), resulting in the production of charged and neutral mesons. Neutral pions decay in photons (observed at Earth as γ-rays), i.e. $\pi^0 \to \gamma\gamma$, while charged pions emit neutrinos in the decay:

$$\pi^+ \to \mu^+ + \nu_\mu, \tag{1.3}$$

with the muon subsequently decaying in: $e^+ + \bar{\nu}_\mu + \nu_e$, and

$$\pi^- \to \mu^- + \bar{\nu}_\mu, \tag{1.4}$$

with the muon subsequently decaying in: $e^- + \nu_\mu + \bar{\nu}_e$. In the $\pi^\pm$ decay chains the three neutrino species are produced with a ratio $\nu_e : \nu_\mu : \nu_\tau = 1 : 2 : 0$, but due to neutrino flavour oscillations during the propagation from the source to Earth, equipartition of the three leptonic flavours ($\nu_e : \nu_\mu : \nu_\tau = 1 : 1 : 1$) is expected at the Earth. The exact source of the high-energy cosmic rays is thus unknown, although some very energetic processes have been proposed as sources or accelerators. An example is represented by Supernova remnants (SNR), i.e. matter ejected at supersonic velocity by the explosion of a Supernova. During this very energetic ending phase of a star, matter can be accelerated to energies greater than 10^{12} eV, resulting in shock waves that move outwards from the centre [6].

Other accelerating processes are represented by Active Galactic Nuclei (AGN), i.e. galaxies with a very bright core of emission in their centre, which are believed to contain a very massive central black hole (10^6 to 10^9 solar masses). Such massive objects are capable of attracting huge quantities of surrounding matter from an accretion disk, and consequently emitting two opposite relativistic jets, i.e. collimated outflows of plasma propagating in perpendicular direction w.r.t the accretion disk. The relativistic jets can accelerate particles to ultra-high energies, thus leading to a very concentrated energy output from a relatively small volume. Early models postulating the hadronic acceleration in the AGN cores, predicted a production of secondary neutrinos [7, 8]. AGNs, as well as Radio Galaxies (galaxies that are very luminous at radio wavelengths) or Galactic clusters can accelerate protons to energies up to 10^{20} eV.

Neutrino astronomy can also provide long baselines for neutrino oscillation studies. Such phenomena, i.e. the neutrino flavour transitions due to quantum mechanical mixing among neutrino flavours (ν_e , ν_μ , ν_τ) and mass eigenstates (ν_1, ν_2, ν_3), have been investigated by several experiments, using solar, atmospheric, reactor or accelerator neutrinos, spanning energies from a fraction of MeV to tens of GeV, providing evidence which contributed knowledge on the subject [6, 9, 10]. Such discoveries imply that neutrinos are massive particles. However, no information on the absolute neutrino masses can be deduced from neutrino oscillation measurement, but they provide estimations of the squared-mass splittings: $\Delta m^2_{ij} = m^2_i - m^2_j, (i, j = 1, 2, 3)$. From the three neutrino mass eigenstates, three mass-squared differences can be constructed, but only two of them are independent: $\Delta m^2_{32} + \Delta m^2_{21} = \Delta m^2_{31}$. Consequently, there are two possible non-equivalent orderings for the mass eigenvalues (the convention is to choose m_1 and m_2 as the mass eigenstate that are close to each other, with $m_1 < m_2$), and the two scenarios are referred to as the normal hierarchy (NH) ($m_1 < m_2 < m_3$) and the inverted hierarchy (IH) ($m_3 < m_1 < m_2$). Some neutrino astronomy experiments are dedicated also to the neutrino oscillation measurements, as will be briefly described in the following sections. Furthermore, neutrinos can provide evidence of Dark matter self-annihilation reactions, which are believed to emit neutrinos [6].

The drawback of using neutrinos as carriers, of course, is that their weak interactions prevent direct detection, and make indirect detection difficult because of their extremely small cross sections (as reported in Section 1.1.2). This implies that very large target masses are needed for the detection, with extremely good background rejection, to observe a measurable flux. The idea of using a very large volume of sea-water as detection medium was proposed by Markov and Zheleznykh in 1960 [11]. The sea water acts in this case simultaneously as the target, the shield and the active detection volume. The proposal foresaw the instrumentation of a large volume of water (as well as ice) with several optical sensors in order to detect the Cherenkov light emitted by the charged particles produced in the CC interaction of neutrinos with water and rock in the proximity of the telescope.

After the pioneering work carried out by the DUMAND collaboration off shore Hawaii Island [12], Baikal was the first collaboration to install a small scale underwater neutrino telescope in the Siberian Lake Baikal (Russia) [13]. On the other hemisphere, the AMANDA detector, constructed at the Amundsen-Scott South Pole Station was completed in 2001 [14]. AMANDA was a first-generation instrument that served as test bench for technologies and as prototype for the km^3-size detector IceCube, which has been taking data in its final configuration since December 2010. In the Northern Hemisphere the largest operating detector is now ANTARES that has a total instrumented volume of about 0.025 km^3, while the construction of KM3NeT, a network of cubic kilometer-scale detectors in the Mediterranean Sea, is currently on-going.

1.1.1 High-energy neutrino detection

The general idea behind the detection of neutrinos in underwater Cherenkov telescopes consists in detecting the optical signal emitted by secondary particles generated in neutrino interactions with the water. This kind of detection is indirect, since the light collected by the photosensors is the Cherenkov radiation emitted by the secondary particles, propagating with a velocity greater than the speed of light in the medium. Muons produced in charged current neutrino interactions represent the *golden channel* for high-energy neutrino telescopes, since they are highly penetrating particles, massive enough ($m_\mu \sim 200\ m_e$) not to lose quickly all the energy via radiative processes (their range in water or rock is several kilometres at $E_\mu \geq 1$ TeV), and carry information on interactions occurring far outside the instrumented volume. However, electron and tau neutrinos can also be identified, although with a worse angular accuracy with respect to the muon neutrinos, through the detection of particle showers[1]. Also neutral current interactions of high energy neutrinos occurring inside the telescope volume can be detected through the measurement of the produced cascades. Such detectors should be as isotropic as possible, in order to be sensitive to events coming from different directions, and should have a volume large enough to efficiently track muons, either produced externally and crossing the detector volume or produced internally (Figure 1.2).

The typical scale of Cherenkov neutrino telescopes is of the order of 1 km^3. Moreover, these detectors have to be shielded from the intense flux of atmospheric muons, originated in the interactions of cosmic rays with the atmosphere. At the Earth surface their flux is about 10^{11} times larger than the one expected for astrophysical neutrino events [15]. Therefore, neutrino telescopes are usually deployed deep underwater. Even at large depths ($\sim$ 3000 m), the flux of muons that reaches the detector is about 6 orders of magnitude more intense than the atmospheric neutrino-induced muon flux [15]. This is the reason why down-going muons cannot be used in the

[1]also known as cascades

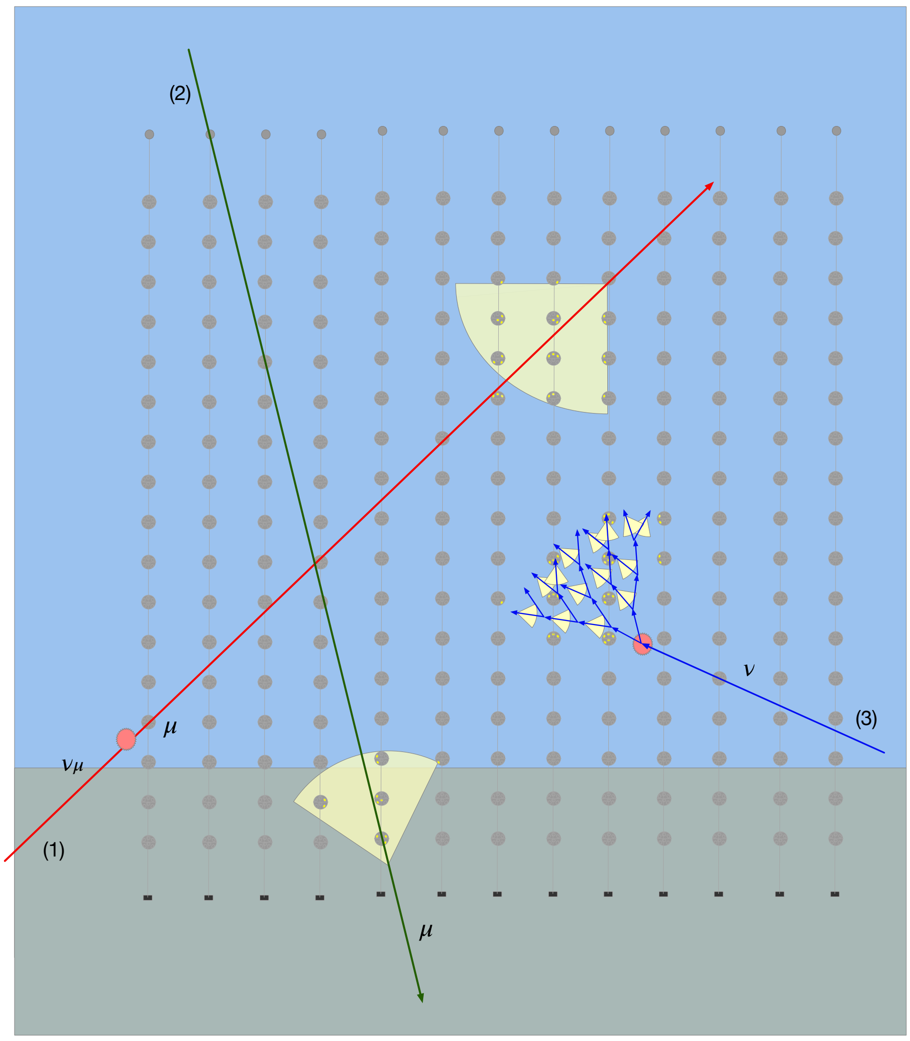


FIGURE 1.2 Representation of 3 types of interaction and Cherenkov light production inside the detector. (1): up-going ν_μCC interaction, generating a muon that propagates into the detector producing a track signature; (2): down-going atmospheric muon generating a track event; (3):shower event, produced either by a ν_eCC interaction or a NC neutrino interaction (any flavour). A schematic representation of the detector geometry is depicted.

search for astrophysical sources, being the signal almost completely overwhelmed by the background. On the contrary, upward-oriented muon events are considered good neutrino candidates, since they cannot be atmospheric: even at the highest energies, in fact, muons are absorbed within a path of about 50 km of water, so they cannot traverse the entire Earth diameter ($\sim$ 13000 km). However, even a downward-looking neutrino telescope suffers the background due to atmospheric muons that are mis-reconstructed as up-going and may contaminate the up-going astrophysical neutrino event sample. A water overburden of thousands of meters (e.g. about 3000 m in the KM3NeT-ARCA case) is therefore required in order to strongly reduce the number of mis-reconstructed down-going events. A completely different source of background is the optical background due to bioluminescence ("bursts" with variable duration from seconds to days, produced by bacteria or small organisms) and to radioactive salts present in sea-water (^{40}K decay), which produce spurious signals on the optical sensors. The reduction of the background is obtained in this case by requiring space-time correlation between signals on nearby photodetectors (see section 2.1).

1.1.2 High-energy neutrino interactions

Neutrino interaction with matter may occur either with a nucleon or an electron. With one exception being the Glashow resonance, occurring at very high energies ($\sim$ PeV) [16], the neutrino-nucleon interaction is dominant for all flavours and energies considered in this work. Neutrinos can interact with nucleons in many ways. The contributing processes are the quasi-elastic scattering (QE), dominant below 1 GeV, the resonance production (RES), mostly significant around a few GeV, and the deep inelastic scattering (DIS), which is dominant for higher energies (i.e. above 100 GeV). The neutrino-nucleon cross sections are shown in Figure 1.3, for neutrinos and antineutrinos respectively, as a function of the neutrino energy, for energies up to a few hundreds GeV [17], while for higher energies, as evaluated by [18], the cross sections are shown in Figure 1.4.

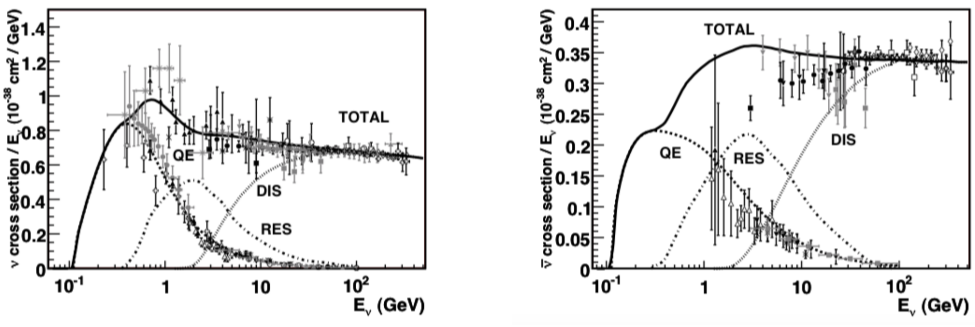


FIGURE 1.3 Neutrino/nucleon (left) and anti-neutrino/Nucleon (right) cross section per energy unit, as a function of neutrino energy up to 10^2 GeV, for different processes, evaluated by [17].

The type of interaction, charged current (CC) or neutral current (NC), as well as the neutrino flavour, characteristically affect the signature of the neutrino events in the detector (Figure 1.5).

The exchanged mediator boson is a $W^{\pm}$ or Z, respectively: the NC interaction only transfers momentum and energy, the CC interaction also transfers charge. Neutrino charged current interactions result in a relativistic charged lepton, with the same

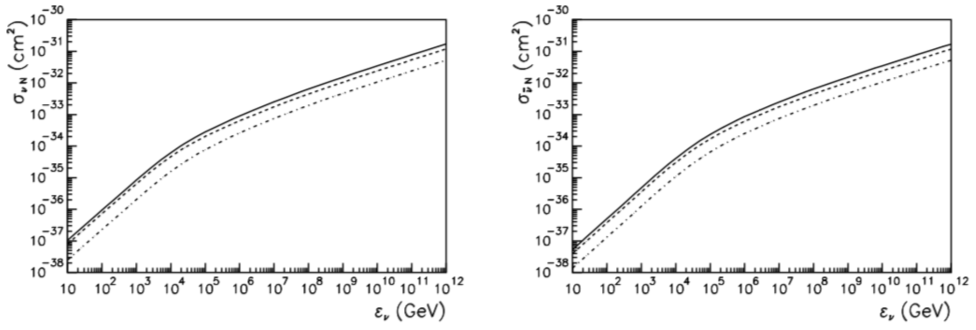


FIGURE 1.4 Neutrino/nucleon (left) and anti-neutrino/Nucleon (right) cross section as a function of neutrino energy up to 10^{12} GeV, taken from [18]. The CC interaction is represented by the dashed line, the NC interactions by the point-dashed line while the solid line shows the total (CC+NC) cross section.

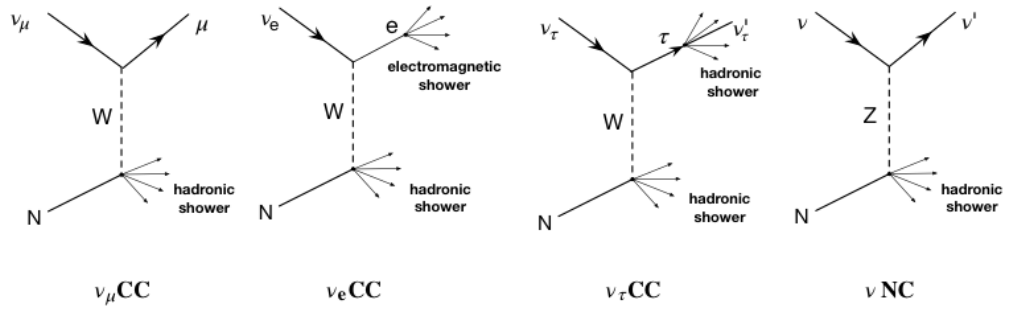


FIGURE 1.5 Representations of the neutrino interactions relevant to KM3NeT. N indicates the nucleon involved in the interaction, while W and Z are the mediator bosons, charged and neutral, respectively. From left to right: ν_μCC interaction, resulting in a muon and a hadronic shower; ν_eCC interaction, resulting in an electron and an electromagnetic plus a hadronic shower; ν_τCC resulting in *double bang* event; νNC interaction, i.e. Neutral Current interaction of a neutrino (any flavour), resulting in a hadronic shower (often producing secondary EM showers in the propagation) and a scattered neutrino.

flavour of the incident neutrino, and a hadronic shower, according to:

$$\nu_l + N \rightarrow l + X, l = e, \mu, \tau \tag{1.5}$$

(and corresponding equation for the anti-neutrinos and anti-leptons respectively). Therefore, ν_μ will produce muons, ν_e will produce electrons and in the latter case an electromagnetic shower will overlap with the hadronic shower. ν_τ events, on the other hand, have a peculiar signature: the τ particle produced in the interaction will travel a certain distance before decaying (as shown in Figure 1.7) according to its energy and, depending on the decay type, might produce a second hadronic shower, resulting in the so-called "double bang event". Neutral Current interactions, on the other hand, are similar for all neutrino flavours. The neutrino interacting with the nucleon will result in a lower energy scattered neutrino and a hadronic shower, as described by:

$$\nu_l + N \to \nu_l + X, l = e, \mu, \tau \tag{1.6}$$

(and corresponding equation for the anti neutrinos). Furthermore, for all hadronic showers, as roughly 1/3 of the produced pions are neutral, there is usually an electromagnetic component that drains off the energy into electromagnetic showers, which overlaps with the hadronic cascade. Therefore, all neutrino interactions that can occur may be classified in (or are a combination of) two signatures: shower-like and track-like events.

A track signature corresponds to a single particle propagating in the medium, producing Cherenkov light as a result of its motion. This behaviour is typical of muons, for the reasons described before, so the track-like events usually indicate a charged current muon neutrino (ν_μCC) interaction. The kinematics of this interaction is such that there is a small angle between the neutrino and the produced muon directions (Figure 1.6).

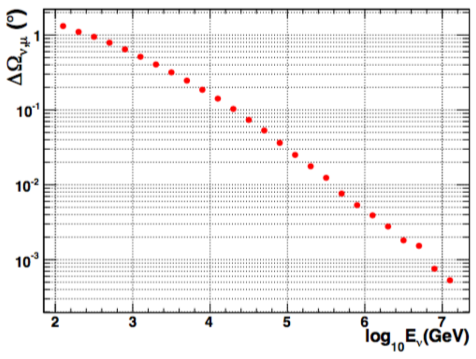


FIGURE 1.6 Median of the angle between the neutrino and the produced muon directions as a function of neutrino energy, as shown in [19].

As only the muon direction can be reconstructed, this sets a limit on the achievable angular resolution. Since the angle decreases with energy, the angular resolution of a neutrino telescope is dominated by the reconstruction above energies of typically a few TeV and by the kinematics of the neutrino interaction below. Muons produced by neutrinos with $E_\nu > 10$ TeV can be considered almost collinear with their parent neutrino [6]. When a high energy muon propagates in a transparent medium (like water or ice), a small fraction of its energy is lost via the Cherenkov radiation emission. The majority of the muon energy loss is due to other mechanisms:

- ionisation of matter, which is the dominant process at energies below 1 TeV;
- radiative losses;

- photo-nuclear interactions.

The total energy loss by muons, neglecting photo-nuclear interactions, for energies above 1 TeV can be expressed as:

$$\frac{dE}{dx} = -\alpha(E) - \beta(E)E \tag{1.7}$$

where the α term is referred to as the ionisation term and the β term is the radiative loss. According to this equation, the muon range is computed as:

$$R = \frac{1}{\beta} ln(1 + \frac{E_\mu \beta}{\alpha}). \tag{1.8}$$

A shower signature, on the other hand, indicates the production of a large number of charged particles, initiated by the decay of a particle or by the interaction of such particle with the surrounding medium. Each particle produced may further decay or interact, resulting in the production of more particles. The process stops when the energy of the particles involved is not sufficient to convert energy into matter. The length of a shower is related to the type and the energy of the initial particle. Electromagnetic showers are typically initiated by the photons emitted in the propagation of high energy electrons (via bremsstrahlung or ionisation). These photons produce electron/positron pairs, leading to a cascading process, since electrons and positrons may further radiate photons via bremsstrahlung reiterating the process. Cherenkov light is emitted by all charged particles with enough energy (above the Cherenkov threshold). The shower evolution is characterised by the radiation length X_0, which is the average distance after which the shower energy is reduced by a factor $1/e$. X_0 in water is approximately 36 cm, while the maximum longitudinal elongation for shower energies of a few GeV is approximately 1 m. [20].

The spatial distributions of electromagnetic and hadronic showers are similar, but in the latter case the process is initiated and driven by nuclear interactions instead of electromagnetic interactions. In water, the nuclear interaction length is about 83 cm [20]. Although hadronic showers are characterised by longer mean free paths, electromagnetic and hadronic showers with the same initial energy have very similar longitudinal apparent length in water, since the masses of the hadrons involved are much heavier (w.r.t. the mass of the electron) and this reduces the quantity of Cherenkov light emitted because the Cherenkov threshold for the involved hadrons is higher. Furthermore, at higher energies the Cherenkov light emitted by hadronic shower approaches that of the electromagnetic ones, and this is due to the fact that 1/3 of the particles produced in the secondary interactions are π^0, which decay mostly in a pair of photons, initiating electromagnetic sub-showers [4].

Neutral Current interactions typically produce hadronic shower events, resulting from the interaction of the neutrinos with the nucleons, and subsequent nucleon fragmentation due to DIS. Electrons have a short mean free path in water, causing an elec-

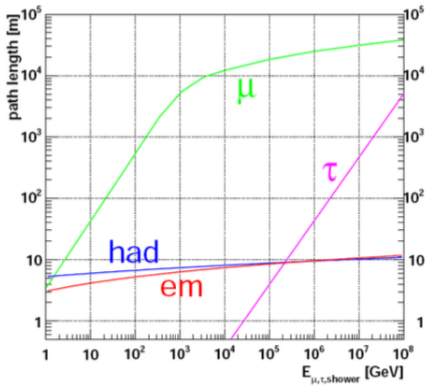


FIGURE 1.7 Path lengths of muons, taus and electromagnetic (em) and hadronic (had) showers in water. Taken from [19].

tromagnetic shower. As a result, the hadronic and electromagnetic showers overlap. Therefore, electron neutrino CC interactions (ν_eCC) have a signature of a shower event similar to NC interactions. Since muons have much longer mean free path than electrons due to their larger mass, which suppresses radiative losses, the muon neutrino CC signature is that of track and/or a hadronic shower. The path lengths of muons and taus as well as those of the hadronic and electromagnetic showers are shown in Figure 1.7. The present work will focus on muon and electron neutrino CC and NC interactions.

1.1.3 The Cherenkov radiation

Charged particles propagating with velocity v in a dispersive medium of refractive index n polarise and excite the surrounding atoms. If v is larger than the speed of light in the medium (c/n), a part of the excitation energy is emitted as coherent radiation, peaked at a characteristic angle θ_C with the direction of the moving particle (Figure 1.8).

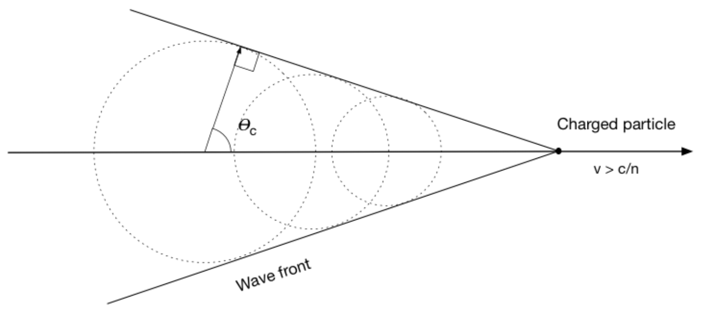


FIGURE 1.8 Cherenkov radiation emitted at a characteristic angle θ_C.

The angle θ_C is linked to n and v by the relation

$$cos(\theta_C) = \frac{1}{\beta n} \tag{1.9}$$

where $\beta = v/c$. The number N of Cherenkov photons emitted by a particle of charge ze per wavelength interval $d\lambda$ and distance travelled dx, is given by:

$$\frac{d^2N}{dxd\lambda} = \frac{2\pi\alpha z^2}{\lambda^2}(1 - \frac{1}{\beta^2 n^2(\lambda)}) \tag{1.10}$$

where λ is the wavelength of the photon and α is the fine structure constant. In the wavelength range from 300 nm to 600 nm, in which water is transparent, this results in about 3.4×10^4 emitted photons per metre of particle track. In this wavelength region, the refractive index of water is $n \sim 1.35$, thus for a highly relativistic particle with $\beta \sim 1$, this leads to a Cherenkov angle θ_C of about 42°. The attenuation of the Cherenkov light in water sets an upper limit to the distance between the optical sensors of the telescope. In order to properly describe the transparency of sea water as a function of the wavelength, it is necessary to introduce the parameters describing absorption and scattering, namely the absorption length λ_{abs} and the scattering length λ_s. Each of these quantities represents the path after which a beam of light of initial intensity I_0 and wavelength λ is reduced in intensity by a factor of $1/e$, through absorption or scattering according to:

$$I_{abs,s}(x) = I_0 e^{(-\frac{x}{\lambda_{abs,s}})}, \tag{1.11}$$

where x is the distance travelled by the photons. The attenuation length is defined as $1/\lambda_{att} = 1/\lambda_{abs} + 1/\lambda_s$. Water is transparent only to a narrow range of wavelengths (350 nm $\leq \lambda \leq$ 550 nm). In particular, λ_{abs} is about 100 m for deep polar ice [21], and it is about 70 m for clear ocean waters [22].

1.2 The KM3NeT experiment

KM3NeT is a multi-km^3 network of neutrino telescopes designed to be installed in the Mediterranean Sea, in a convenient location to look for high-energy neutrino sources in the inner part of our Galaxy. Three sites (40 km off-shore Toulon, France, at a depth of 2500 m; 80 km off-shore Capo Passero, Italy, at a depth of 3500 m; 20 km off-shore Pylos, Greece at depths of 3500-5000 m) have been chosen to host the telescopes, and the first two, namely the Italian and the French sites, are already hosting working detectors. The main goals of KM3NeT are to identify sources of cosmic neutrinos and to establish the neutrino mass hierarchy (see Section 1.1). Since neutrinos propagate undisturbed from their sources to the Earth, even modest numbers of detected neutrinos can be of great scientific relevance, e.g. by indicating the astrophysical objects in which cosmic rays are accelerated, or pointing to places where dark matter particles annihilate or decay. The analyses performed in large ice and water Cherenkov detectors rely upon the reconstruction of the muon direction and energy (in ν_μCC events), to get an estimate of the direction and energy of the neutrino. As the two studies are conducted with neutrinos having very different energy scales, two different detector configurations have been designed, namely: ARCA (Astroparticle Research with Cosmics in the Abyss), the high energy detector; and ORCA (Oscillation Research with Cosmics in the Abyss), dedicated to the low energy neutrino detection. In the latter, since low energy neutrinos are more abundant and a denser configuration is needed for a reliable reconstruction, a smaller detector volume is constructed. The structure of the two configurations is basically the same, the main difference being in the spacing of the components. The whole KM3NeT detector will be composed by three so-called *building blocks*, of which two will be dedicated to the high energy measurements (ARCA), while one will be used for ORCA. Currently, KM3NeT phase-1 is under construction, which will have 24 ARCA Detection Units and 7 ORCA Detection Units deployed in the Italian and French sites, respectively. Each building block will contain a set of 115 Detection Units, each in the shape of a string of 18 Digital Optical Modules (DOMs). The Digital Optical Module (DOM) [23] is a transparent 17 inch diameter glass sphere made of two separate hemispheres, housing 31 Photo-Multiplier Tubes (PMT) and their associated readout electronics (Figure 1.9). The design of the DOM has several advantages over traditional optical modules using single large PMTs, as it houses three to four times the photo-cathode area in a single sphere and has a much more uniform angular coverage [6]. As the photo-cathode is segmented, the identification of more than one photon arriving at the DOM can be done with high efficiency and purity. In addition, the directional information provides improved rejection of optical background. Within the Optical Module, the PMTs are arranged in 5 rings of 6 PMTs plus a single PMT at the bottom, pointing vertically downwards. The PMTs are spaced at 60° in azimuth and successive rings are staggered by 30°. There are 19 PMTs in the lower hemisphere and 12 PMTs in the upper hemisphere, held in place by a mechanical support [6].

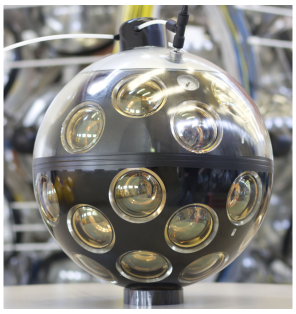


FIGURE 1.9 KM3NeT Digital Optical Module (DOM) [24].

For KM3NeT/ARCA, each string is about 700 m in height, with DOMs spaced approximately 36 m apart in the vertical direction, starting about 80 m from the sea bed. For KM3NeT/ORCA, each string is 200 m in height with DOMs spaced 9 m apart in the vertical direction, starting about 40 m from the sea floor. The lower end of each string is anchored to the sea bed and connected to the shore station with an electro-optical cable, which transfers data to shore via optical fibres. The upper part of the detector reaches a depth of about 2700 m in the ARCA case under the sea level and is held approximately vertical by a submerged buoy, to reduce the horizontal displacement of the top relative to the base, due to large sea currents. This work will focus on the ARCA detector, and in particular, all the simulations used have been based on one ARCA building block. Therefore, all of the analyses shown will be following this configuration. The KM3NeT-ARCA design has been carefully optimised to maximise the sensitivity to the Galactic sources. One of the findings in this process is that the overall sensitivity is not reduced if the neutrino telescope is split into separate building blocks, provided they are large enough, at least 0.5 km^3 each [6].

1.3 Neutrino interaction events in KM3NeT

Event simulations and detector response

Monte Carlo simulation tools are necessary for understanding the systematic effects in a detector and to determine the detection efficiency of the signal of interest and the background. The muon and electron neutrino charged and neutral current interaction events considered in this study have been simulated with software that has mostly been developed in the ANTARES Collaboration and then adapted to the KM3NeT configuration. Muon and electron neutrino Charged Current interactions will be referred to as ν_μCC and ν_eCC, while the Neutral Current interactions will be denoted as ν_μNC and ν_eNC respectively. In event generation, the interaction of a neutrino and the passage of a particle through the detector are simulated. The simulation is based on the nominal detector geometry described in the previous section. Each of the two ARCA blocks is treated identically and independent simulations are performed for a single block, and the effective lifetime (event rate) is multiplied by two. The first step of the simulation chain is the generation of particle fluxes in the so-called generation volume, which is significantly larger than the instrumented volume, in order to include particles that can travel larger distances, e.g. the high energy muons. To save computing time, a region containing the instrumented volume at its centre is defined (the so-called *can*), according to which the simulated events are recorded (Figure 1.10). The can typically extends three times the absorption length of light in water around the instrumented volume. For each generated event, the produced particles are prop-

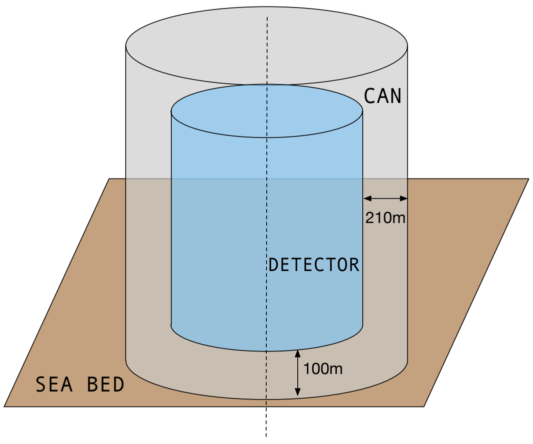


FIGURE 1.10 Detector and Can

agated and if at least one reaches the can, the event is recorded. Each neutrino interaction is simulated using the GENHEN code [25], in which an energy spectrum according to a power law with a given index $E^{-\gamma}$ is assumed. The simulation includes propagation through Earth, deep-inelastic scattering, quasi-elastic scattering and resonant interactions. Atmospheric muons are simulated using the MUPAGE code [26]. MUPAGE uses a parameterisation of the atmospheric muon flux at different energies and zenith angles. As a result, events can be relatively easily generated with sufficient statistics, to match the actual detection rate of atmospheric muons. In addition to single atmospheric muons, atmospheric muon bundles are simulated: i.e. groups of muons produced in the core of an extended air shower (EAS). Such events exhibit a signature very similar to that of a single high-energy muon in the detector, but with a stochastic energy-loss pattern that is much more uniform, and a non-negligible lateral spread (w.r.t. the characteristic spatial resolution scale of ARCA) [6]. Muon bundles are especially important at high energies. Since the energy spectrum of atmospheric muons is steep, MUPAGE simulations have been performed for different muon energy bins. This allows simulating adequate statistics at energy regions of interest for cosmic neutrino detection. The light production for showers is simulated using a so-called "multi-particle" approximation. Except for muons and taus, the light production of particles is simulated by treating them like an electron shower with a scaled equivalent energy and distance to the vertex. The light production of particles in the can is simulated using tabulated results from a `GEANT 3.21` simulation. This approach is used to reduce the computing time that would be required by single-photon tracking. The code used for this simulation is called KM3 [27]. This software projects the light produced to the PMTs in the detector, taking into account the scattering and absorption of the light in the sea water. In the next step the light detection is modelled including absorption in the glass sphere and the gel, as well as the PMT efficiency and PMT angular acceptance. Light from ^{40}K decays and dark rates from the PMTs are simulated by adding 5 kHz of random noise to the signal reaching the PMTs. Light from bioluminescence bursts is not simulated. The PMT response is simulated using the `JPP` software framework [28]. It includes the transition time and the amplification of the PMT as well as the effects of the readout electronics. The detector response, as well as some packages contained in `JPP` will be described in the trigger optimisation section (2.1). Since the detector response is recorded as a collection of hits, i.e. signal detected by the PMTs, and times, the shower-like and track-like topologies produce very different space-time hit patterns in the detector. In particular, the shower-like events are characterised by a very dense distribution of hits, close to the neutrino interaction point. A significant fraction of the neutrino energy is released in a hadronic shower (and, in the case of ν_eCC interactions, the rest in an electromagnetic cascade), thus allowing for a good estimate of the neutrino energy. A track-like event is characterised by the Cherenkov light from the emerging muon that can travel large distances through Earth rock and sea water. The spatial hit pattern in this case is closely related to the muon direction, thus allowing for a precise measurement of the latter. Starting from the ANTARES experience, algorithms that reconstruct direction, energy and interaction vertex of the neutrinos from the muon tracks or the showers have been developed. These have been optimised for *pure* track events (ν_μCC events far from the detector, where only a single energetic muon is observed) and for shower events (NC interactions of all neutrino flavours and ν_eCC events, where only a shower is observed), respectively.

Chapter 2

Event triggering in KM3NeT

Triggers have paramount importance during data acquisition and analysis to select events and reduce the background contamination. Most analyses, such as the event reconstruction algorithms or Neural Network-based particle identification, are affected by different choices of the trigger conditions, as they rely on event selection. For instance, loose trigger conditions may result in a large amount of data collected, which can be counter-productive for the computing time and memory occupation; on the other hand too strict cuts may cause an excessive loss of data, causing inefficiencies. The solution is finding a balance between the two extreme options, to get the best from the trigger conditions applied both on-line and off-line. This chapter will describe a study performed on the trigger conditions, by investigating how tuning trigger parameters affects the data selection. The main goal of this study is to ensure that the best trigger conditions are used to select the events further fed into Deep Learning models, as reported in Chapter 4.

2.1 Online and offline triggers in KM3NeT

The readout of the KM3NeT detector is based on the “all-data-to-shore” concept, according to which all the analogue signals from the PMTs exceeding a threshold (typically voltage equivalent to 0.3 photo-electrons) are digitised and sent to the shore station to be processed. Dedicated software filters possible physics events from background. Each analogue signal is converted to digital by an individual time-to-digital converter (TDC) implemented in an FPGA (Field Programmable Gate Array) on the Central Logic Board (CLB), contained in the DOM [29]. Converted data are then transferred to shore via an Ethernet network of optical fibres. Hits are recorded as a pair of the start time and the duration of the signal above a predefined threshold (also called *time-over-threshold*); the time during which the signal is above the threshold is then recorded, and is referred to as the "time-over-threshold" quantity (ToT) (Figure 2.1).

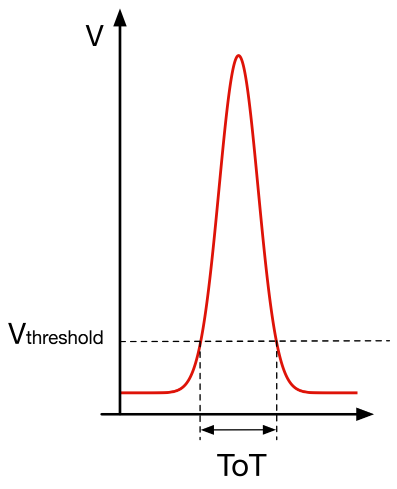


FIGURE 2.1 Schematic view of the Time-over-Threshold technique: the duration of the time interval in which the signal exceeds a threshold value is recorded as ToT. Reference: [30].

The raw data are organised in segments called *timeslices* of 100 ms duration. Each timeslice contains a data frame per DOM in which the information of the hits (start time and ToT) is stored. The start time of the hit is recorded with 1 ns granularity. The hit time is determined with respect to the internal clock of all PMTs of every DOM. Thus, every pulse, along with its *time over threshold* and time of the hit, is commonly referred to as a *hit*, with the time granularity of 1 ns. As a result, the total data rate for a single building block is about 25 Gb/s. A reduction of the data rate is required to store the filtered data on disk [6]. The following selection criteria, to reduce the amount of background events recorded, are applied to the data on shore, except the first selection, which is applied off shore:

- the level-zero filter (L0) is the threshold for the analogue pulses that are sent to the shore station;
- the level-one filter (L1) refers to a coincidence of two or more L0 hits from different PMTs in the same optical module, within a fixed time window. The scattering of light in deep-sea water is such that the time window can be very small. A typical value is $\Delta T = 10$ ns. The estimated L1 rate per optical module is then about 1000 Hz, of which about 600 Hz is due to coincidences from ^{40}K decays (in the ARCA environment);

- The remaining part arises from random coincidences that can be reduced by about a factor two relying on the known orientations of the PMTs. This is referred to as the level-two filter (L2).

In order to reject random background, space-time coincidences between the hits are required. The basic idea of all trigger algorithms is that events produce clusters of hits that are separated in time by light propagation in water, while background light is uncorrelated. In particular, first a selection of all of the local coincidences is performed (L1 coincidences, within the same DOM in the 10 ns time window), then clusters of hits are selected, so that any hit in the cluster is related to the remaining ones, according to the following causality relation:

$$|\Delta t|< |\Delta r|/c_{water} + T_{extra} \tag{2.1}$$

where Δt is the time difference between the two hits, Δr is the distance between the two considered hits, c_{water} is the group velocity of the hits in water, i.e. $\frac{c}{n}$, and T_{extra} is an additional time window, considered in the event to account for timing uncertainties, as well as photon scattering, and is typically set to about 10-20 ns. After the cluster of hits is defined, it is further extended by including all the causally connected hits for which the following conditions are fulfilled:

- the hits are causally connected to at least 75% of the cluster hits;
- the hits are closer than 50 m to at least 40% of the cluster hits;

Separate trigger algorithms operate in parallel on the data, each optimised for a specific event topology: track-like and shower-like events (Section 1.1.2). Therefore, a *Muon trigger* and a *Shower trigger* are defined for each topology, respectively.

Muon trigger

The general idea to trigger a muon track event requires the assumption of a muon direction. In this way, the causality relation can be rewritten as follows [19, 31]. Figure 2.2 shows the relation between a propagating muon and the arrival time of the Cherenkov light at a given PMT, which can be expressed as:

$$t_i = t_0 + \frac{z_i - z_0}{c} + tan(\theta_C) r_i \frac{n}{c}, \tag{2.2}$$

where t_i is the time of the hit, i.e. the time at which the photon reaches the PMT; t_0 is the starting time of the event, i.e. the instant at which the muon is in the

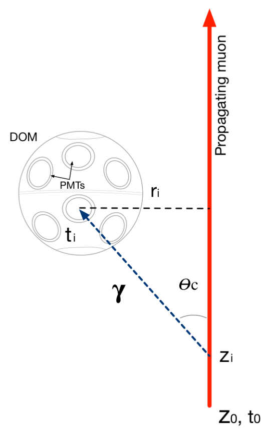


FIGURE 2.2 Schematic view of a muon (solid arrow) propagating in the vicinity of a DOM; Cherenkov light (dashed arrow) emitted and collected by a PMT.

position z_0; z_i is the position of the particle when the Cherenkov photon is emitted at the Cherenkov angle θ_C; r_i is the distance of the PMT from the direction of the muon. This relation is evaluated after having rotated the coordinate system so that the muon is seen travelling along the Z axis. Thus, the time difference of two hits in this coordinate system satisfies the inequality:

$$|(t_i - t_j)c - (z_i - z_j)| \leq \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} tan(\theta_C), \tag{2.3}$$

where t_i and t_j are the times (of arrival at the PMTs) of the hits, and x, y, z the corresponding coordinates [31].

For each assumed direction, the intersection of a cylinder (whose axis is the muon direction) with the 3D array of optical modules is considered, and all the PMTs falling inside the intersection are included in the event (Figure 2.3). The diameter of this cylinder (also known as *road width*) corresponds to the maximal distance travelled by light in sea water, and is safely set to a few times the absorption length [6].

With a requirement of four (or more) L1 hits, this filter shows a very small contribution of random coincidences: such constraints improve the signal-to noise ratio (S/N) of an L1 hit by a factor of (at least) 10^4 compared to the general causality relation [6].

Shower trigger

For shower events, a point-like emission is assumed. As a consequence, the definition of a maximum photon travel distance limits the maximum distance *Dmax* between hit DOMs. The maximal allowed time difference ΔT between two causally connected

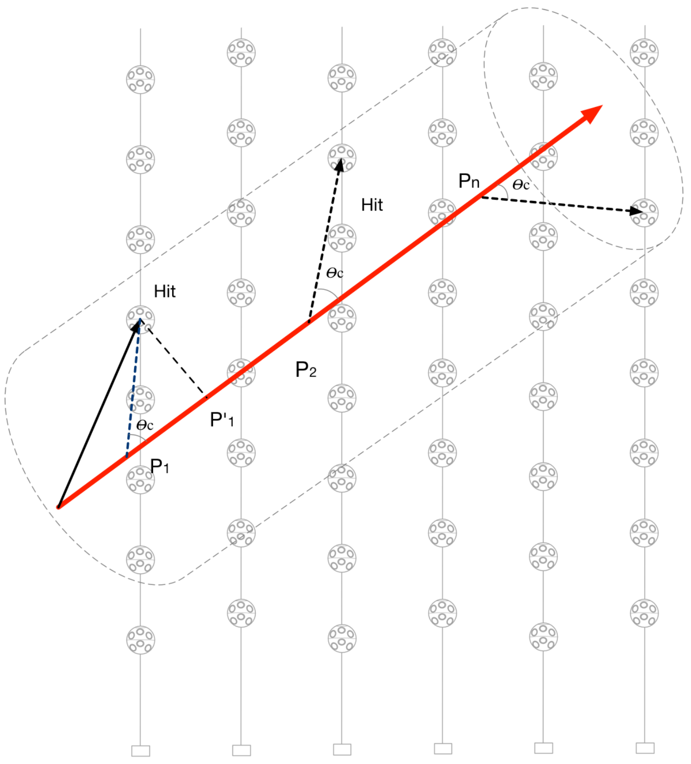


FIGURE 2.3 The geometry of the detection of the Cherenkov light emitted by a propagating muon (solid arrow). The dashed arrows represent the Cherenkov light emitted with the characteristic angle. A cylinder is superimposed to represent the *road width* defined for hits selection.

hits separated by a distance d is then:

$$\Delta T_d = \begin{cases} d/c_{water} + T_{extra} & \text{if d} < D_{max}/2 \\ (D_{max} - d)/c_{water} + T_{extra} & \text{if } d > D_{max}/2 \end{cases} \tag{2.4}$$

Such filters help reducing the total number of PMTs to be considered in each event and the width of the time window obtained with the causality relation [6]. It is worth noting that the number of computers and the speed of the algorithms determines the performance of the system and hence needs to be properly sized to avoid affecting the physics output of KM3NeT.

2.2 Trigger parameters

The response of the detector is simulated by the tools named `JTriggerEfficiency` and `JTriggerProcessor`, contained in the `JPP` framework - the official simulation and analysis framework developed in the KM3NeT Collaboration. The main difference between the two programs is that `JTriggerProcessor`, often referred to as `JTP`, is designed to trigger DAQ data files (i.e. the real data events), while `JTriggerEfficiency` (`JTE`) is to be applied to Monte Carlo simulations, with the added functionality of simulating uncorrelated background hits, usually discarded by the space-time coincidence requirements. The trigger algorithms implemented in `JTE` include both the muon and the shower trigger. The shower trigger is used for the optimisation of the detection of shower-like events, which are produced by charged-current (CC) interactions of electron (anti)neutrinos ν_e ($\bar{\nu}_e$), and ν_μ , ν_e , ν_τ ($\bar{\nu}_\mu$, $\bar{\nu}_e$, $\bar{\nu}_\tau$) neutral current (NC) interactions, resulting in particle showers, as explained in 1.1.2: the produced electron gives rise to an electromagnetic shower, while the hadronic system can develop a hadronic shower, according to its energy and to the decay modes of the particles produced. For events induced by the neutral current (NC) interaction of neutrinos, both electromagnetic and hadronic showers can be produced. On the other hand, track-like events are produced mainly by ν_μ charged current (CC) interactions and high energy atmospheric muons. Depending on which trigger selects a hit, `3DShowerTrigger` or `3DMuonTrigger`, each hit has an associated *trigger bit* set for analysis purposes. If the hit is triggered by both algorithms, both trigger bits are set.

2.2.1 JTriggerEfficiency

All the trigger parameters can be selected via command line arguments when running `JTriggerEfficiency` (or `JTriggerProcessor`). Default values are hard-coded in the software, but each parameter can be tuned. The following table shows the trigger parameters and their default values.

Besides these tunable parameters, further options regarding the output settings can be used (e.g. whether to save only triggered hits, or verbosity options). The software is typically run by specifying the *detector file*, containing a detailed description of the detector geometry to use in the analysis, the input event file and the output file name. In the following, the tunable parameters will be described in detail.

Trigger parameter	Default value
`trigger3DShower.enabled`	0
`trigger3DShower.numberOfHits`	5
`trigger3DShower.DMax_m`	250
`trigger3DShower.TMaxExtra_ns`	20
`trigger3DShower.factoryLimit`	100
`trigger3DMuon.enabled`	0
`trigger3DMuon.numberOfHits`	5
`trigger3DShower.TMaxExtra_ns`	20
`trigger3DMuon.DMax_m`	1000
`trigger3DMuon.roadWidth_m`	120
`trigger3DMuon.gridAngle_deg`	10
`trigger3DMuon.TMaxExtra_ns`	20
`trigger3DMuon.factoryLimit`	100
`L2Min`	2
`ctMin`	-1
`TMaxLocal_ns`	10
`TMaxEvent_ns`	5000
`numberOfBins`	1000
`combineL1`	1

TABLE 2.1 Trigger parameters and their default values

2.2.2 JTE parameters tuned

The trigger parameters that can be tuned and passed to `JTE` and `JTP` via command line arguments are:

- `trigger3DShower(trigger3DMuon).numberOfHits`: the minimum number of L1 hits required for an event in the coincidence time window to be triggered. This parameter can be set separately for the muon and the shower triggers;
- `trigger3DShower.DMax_m`: the maximum distance in m between PMTs to be used for triggered event time;
- `TMaxExtra_ns`: maximum time in ns added to the coincidence time window to select the L1 hits contributing to a triggered event. This time is added to the maximum event time (see Section 2.1);
- `trigger3DMuon.roadWidth_m`: radius of the cylinder surrounding a muon track that selects which PMTs are considered in the event;
- `combineL1`: if activated, multiple L1 hits occurring on a single DOM in the local time window are merged to create a single L1 hit per DOM;
- `TMaxLocal_ns`: local time window according to which L1 coincidences are defined;

- `B` (string): random noise rate to add to the input signal, to take into account random coincidences due to ^{40}K decay. A common choice for this is: `"5000, 500, 50, 5, 0.5"` (Hz), to include 2, 3, 4, 5-fold coincidences per DOM and stay on the safe side with 5000 Hz of simulated background rate;
- `gridAngle_deg`: angular step considered in the scanning of the sky: the number of directions for which a track hypothesis is tested defines the maximum number of directions to test;
- `factoryLimit`: cut on maximum size of hits in the event; if exceeded, the event is written in any case;
- `L2Min`: minimum number of L2 coincidences required for an event to be triggered;
- `ctMin`: minimum value for the cosine of the zenith angle. If -1, then the down-going events are included, whereas setting this parameter to 0 will consider only the horizontal and up-going events.

The trigger algorithm works as follows: the time difference between the first and the last L1 hit of each event is evaluated and compared to the local coincidence time window. If in this event time window the number of correlated L1 hits is larger than `numberOfHits`, then the event is triggered. This chapter concentrates on the effects of the tuning of the following parameters on the trigger efficiency: the number of L1 hits (`numberOfHits`), the time window `TMaxExtra_ns`, the distance parameters `roadWidth_m` and `DMax_m` for muon and shower trigger respectively, and the boolean parameter `combineL1`. In the following, if not explicitly indicated the parameters `numberOfHits` and `TMaxExtra_ns` are set for both the muon and shower trigger at the same value.

`combineL1=0` vs. `combineL1=1`

Events in KM3NeT can produce multiple hits on the same DOM, if these hits are collected by several PMTs in the local time window. In principle, each hit occurring in the local time window counts as one. Thus, an event with 5 or more hits in the same DOM could in principle be triggered. The parameter `combineL1`, if activated, merges the multiple hits occurring on a DOM in the local time window, resulting in a single L1 hit on the selected DOM. This way, multiple DOMs must be triggered in order to let the event pass the selecting conditions. Deactivating this parameter, a larger number of hits is collected. Consequently, if the bare number of hits is considered to discriminate the events, a greater number of triggered events is obtained, especially for lower energies. In Figure 2.4 the trigger efficiency is compared for the default trigger conditions (referred to as `std_trigger_combineL1`), and for the standard trigger with the parameter deactivated (referred to as `std_trigger_nocomb`). As expected, the resulting trigger efficiency would be much higher with respect to the case in which the parameter is on. On the other hand, this choice dramatically increases the rate of background signal. As an example, for the ^{40}K decay, Table 2.2 shows the measured rate as a function of the simulated random background rate, for the two different choices of the trigger parameter `combineL1`. The typical rate of the ^{40}K signal is of about 5 kHz. Nevertheless, it is possible to detect bursts of higher rates, for which the trigger rate must be kept low. Furthermore, it is important to notice that multiple hits occurring on the same optical module within a larger timespan can still count as separate hits. For this reason, additional conditions can be applied.

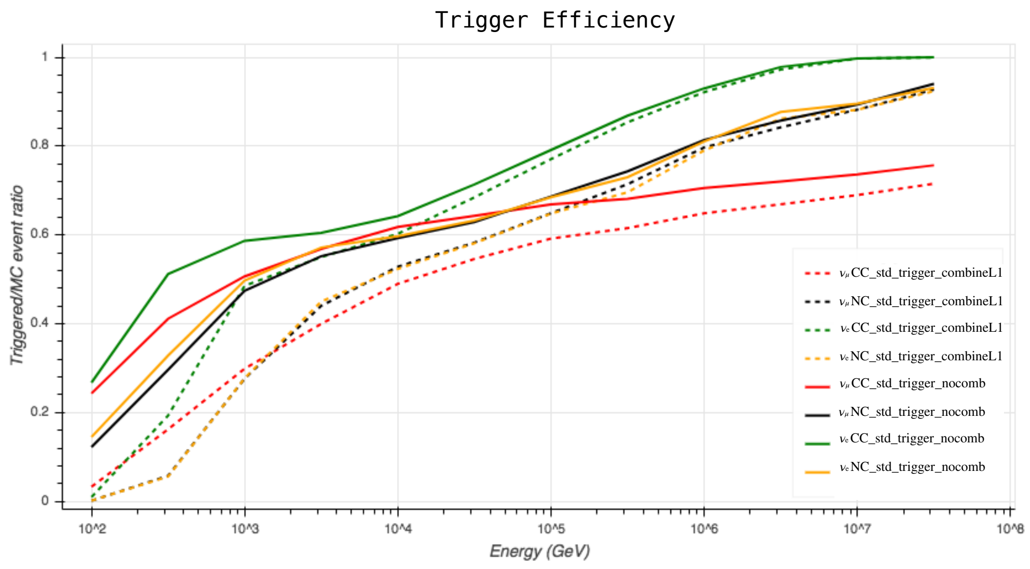


FIGURE 2.4 Trigger efficiency for standard trigger as a function of the neutrino energy, for different choices of the parameter `combineL1`

Input signal (kHz)	Trig. rate (Hz) w/ `combineL1=1`	Trig. rate (Hz) w/ `combineL1=0`
5	0	$1.5 \cdot 10^3$
10	0.9	$2.2 \cdot 10^3$
15	9.8	$3.5 \cdot 10^3$
20	$1.1 \cdot 10^2$	$6.4 \cdot 10^3$

TABLE 2.2 Table of comparison of the background rate for default trigger conditions with `combineL1=0` and `combineL1=1` for different rates of input signal.

Number of hits vs. number of DOMs hit

The parameter `trigger3DShower(trigger3DMuon).numberOfHits`, for both shower and muon triggers, has been tested with different configurations. The tested values are 4 and 5, since lower values would represent an extremely loose condition, whereas higher values would be too strict a constraint. Even though L1 hits are merged locally on a single optical module, multiple hits are still possible if occurring in a time interval that is larger than the local time window (`TMaxLocal_ns`). The number of DOMs hit, instead of the bare number of hits, proves itself to be more effective in the selection of events. This choice allows to reject events with multiple hits on the same DOM, but few DOMs hit. This selection improves the quality of event detection, even though it decreases the trigger efficiency, as can be seen in Figure 2.5. In the following sections and in all of the presented analyses, the term "number of hits" will be referred to as the minimum number of hits occurring on different DOMs per event.

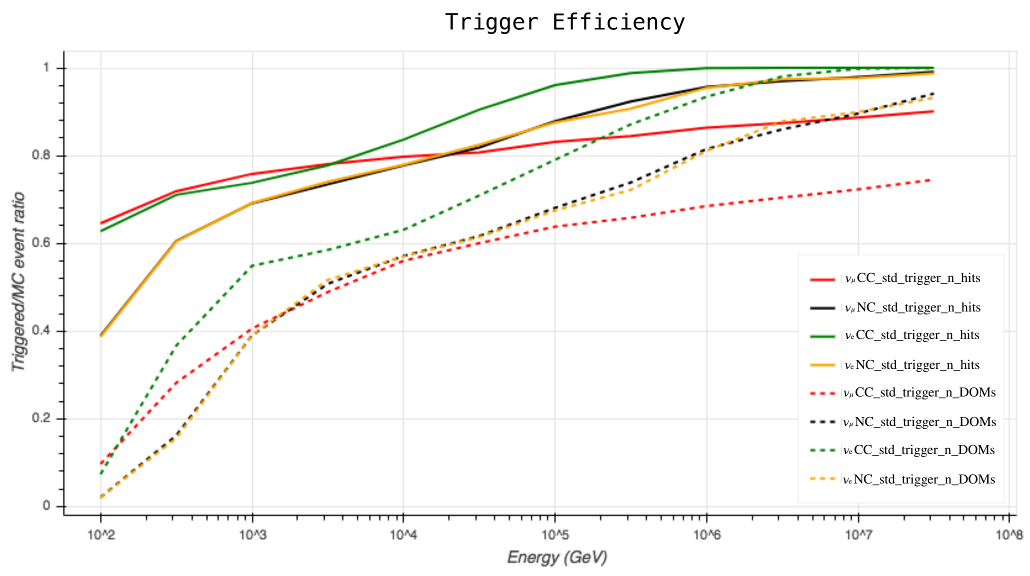


FIGURE 2.5 Trigger efficiency as a function of the neutrino energy for `numberOfHits=5` vs. trigger efficiency selecting events with more than 5 DOMs hit.

2.3 Trigger efficiency optimisation

Different selections of the trigger parameters have been tested, in order to maximise the number of collected events, and to keep the background contamination low at the same time. Several combinations of trigger parameters have been considered, and trigger efficiency has been evaluated and compared for each choice. A sample of approximately 500,000 events from the official KM3NeT Monte Carlo production have been processed with `JTriggerEfficiency` for each trigger configuration. The neutrino flavour and interactions that have been considered are: charged and neutral current muon neutrino interactions and charged and neutral current electron neutrino interactions, often referred to as ν_μCC , ν_μNC , ν_eCC and ν_eNC respectively. For each configuration, the number of triggered events have been collected and compared to the number of simulated events, i.e. the total simulated events before the simulation of the detector response via `JTE`. The trigger efficiency, namely the ratio of triggered over simulated events per energy bin, is defined as:

$$Eff = \frac{number\ of\ triggered\ events}{number\ of\ simulated\ events} \tag{2.5}$$

The efficiency has been evaluated as a function of the energy of the simulated particles at *can* level (i.e. the energy of the simulated particles that reach the can). For each energy bin (8 energy bins in a logarithmic scale from 10^2 to 10^8 GeV), the number of triggered/simulated events is calculated, and the obtained curve is shown for every different selection of trigger parameters. In particular, `numberOfHits` ranging from 4 to 5 has been combined to increasing `TMaxExtra_ns` from 20 ns to 370 ns. Relaxing the request on the minimum number of hits required for an event to be triggered, as well as enlarging the time window, allows more events to be collected, as might be expected. As this can result in a larger amount of background detected, for every investigated set of parameters, the background rate is evaluated alongside the efficiency in order to choose the combination that maximises it with a sustainable background rate of approximately $100-150$ Hz (set on the basis of disk/network speed occupancy).

2.3.1 Standard trigger set vs. alternative set

A first comparison has been carried out considering the default set of trigger parameters in `JTE` and a selection of values for the same parameters that have been studied in the master thesis reported in [32]. This work focussed on investigating the possibility to optimise trigger conditions, while ensuring atmospheric muons background rejection. In the following, the two lists of parameters considered in the comparison will be referred to as **"std trigger"** for the standard trigger conditions, i.e. the default parameters, and **"alter"** for the alternative set, as defined in [32]. The values for the two sets of trigger parameters are reported in Table 2.3.

Differently from the standard trigger conditions, the **alter** parameter set relaxes the selection on the number of hits to 4 (i.e. `numberOfHits`), while the distances (i.e.

Parameters	**std**	**alter**
`trigger3DShower.numberOfHits`	5	4
`trigger3DShower.DMax_m`	250	130
`trigger3DShower.TMaxExtra_ns`	20	370
`trigger3DMuon.numberOfHits`	5	4
`trigger3DMuon.roadWidth_m`	120	50
`trigger3DMuon.TMaxExtra_ns`	20	150

TABLE 2.3 Comparison of the **std** and the **alter** parameter sets. The standard values are the default parameters set in JTE, as reported in table 2.1, while the alternative set has been proposed by [32].

`DMax_m` for the shower trigger, and `roadWidth_m` for the muon trigger) have been considerably reduced, allowing for more triggered events, but with fewer DOMs taking part in the process. On the other hand, very large time windows (i.e. `TMaxExtra_ns`) have been defined. This parameters selection relies on allowing more coincidences on neighbour DOMs, thus including more hits collected in the large time windows, but reducing the number of total DOMs considered, by restricting the distances allowed.

The trigger efficiency has been evaluated for electron and muon neutrino charged current and neutral current interactions for the two sets of trigger conditions and is shown in Figure 2.6.

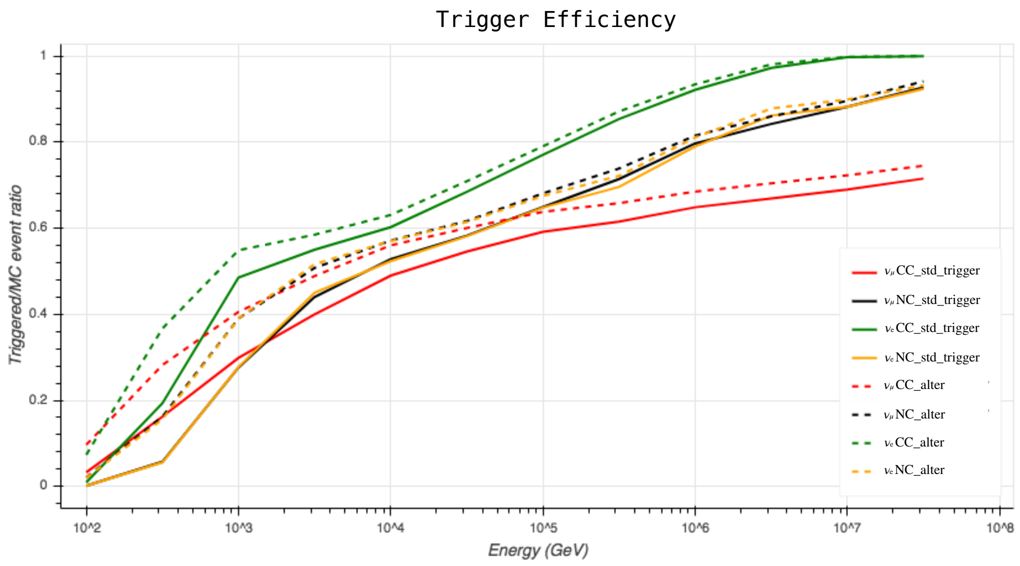


FIGURE 2.6 Trigger efficiency vs. neutrino energy for **std trigger** conditions compared to **alter** conditions.

Trigger efficiency, in terms of number of events that satisfy the trigger requirements, is higher in the **alter** case, and the effect is more evident for lower energies ($E < 10$ TeV).

Such effect may be related to the fact that with such large time windows more hits are collected, even if they are not caused by the propagating particle.

However, to keep the background rate under control, the effects of both sets of trigger parameters on the rate of random background collected have been compared as well. A random background signal, accounting for random coincidences due to ^{40}K decays, has been simulated with the tool `JRandomTimesliceWriter` and processed with `JTriggerProcessor`. As can be seen in Table 2.4, for the `alter` parameters set the collected rate is too high to be handled: in fact, for some choices of the input rate it exceeds 1 kHz.

Input signal(khz)	**std trigger rate (Hz)**	**alter trigger rate (Hz)**
5	0	3.8
10	0.9	24.3
15	9.8	$2.0 \cdot 10^2$
20	$1.1 \cdot 10^2$	$1.2 \cdot 10^3$

TABLE 2.4 Table of comparison of the background rate for **st** and **alter** conditions.

2.3.2 Dependency on the time window

From the previous comparison, it looks inconvenient to enlarge the time window and reduce the minimum number of hits at the same time. On the other hand, it is worth to further investigate the stricter requirement on the distances introduced in [32]. To this aim, the dependency on the time window of the trigger efficiency has been evaluated, fixing, at the same time, the `roadWidth_m` and the `DMax_m` as in the **alter** conditions. Figure 2.7 shows the dependency of the trigger efficiency on the time window for several (increasing) values: in this analysis, the standard values are compared to the **alter** set, and to 4 different versions of the latter, with `TMaxExtra_ns` varying in the following set of values: 20 ns, 50 ns, 100 ns, 150 ns (e.g. keeping the parameters as set in the **alter** conditions, and varying the extra time window only).

As can be seen in more detail in Figure 2.8, the **alter** set with a 20 ns time window slightly increases the trigger efficiency in the low energy region (i.e. below 10 TeV), with respect to the standard conditions, while the curve does not change for high energies. For larger time windows, instead, the change in efficiency is evident over the whole energy spectrum. Moreover, there appears to be a saturation effect above 100 ns: the relative change in efficiency is small, thus setting an upper limit to the `TMaxExtra_ns` time window.

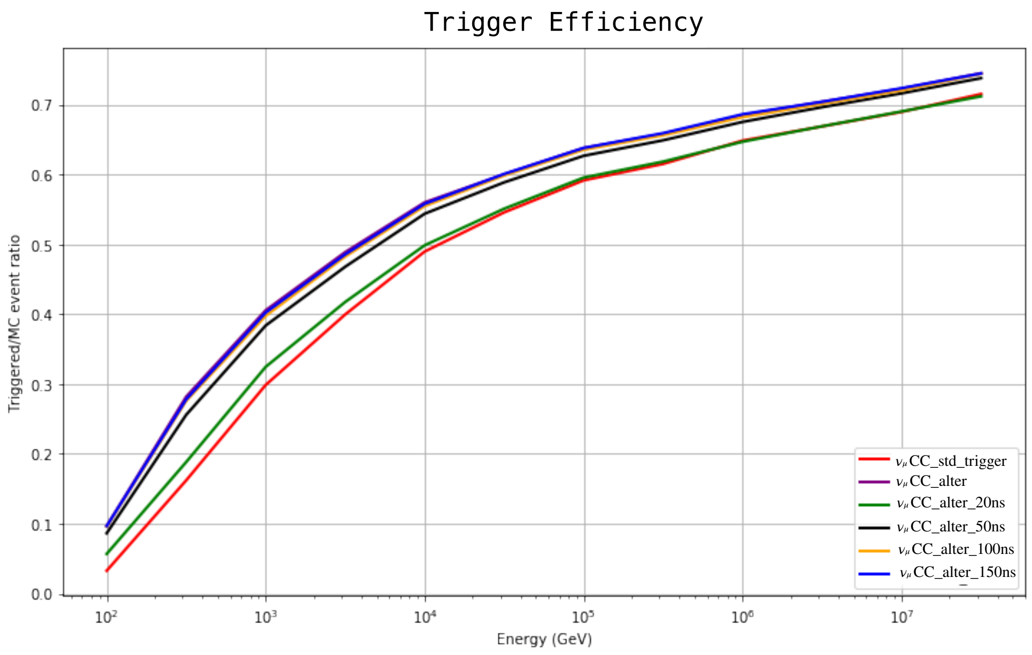


FIGURE 2.7 Trigger efficiency (vs. neutrino energy) dependency on time window.

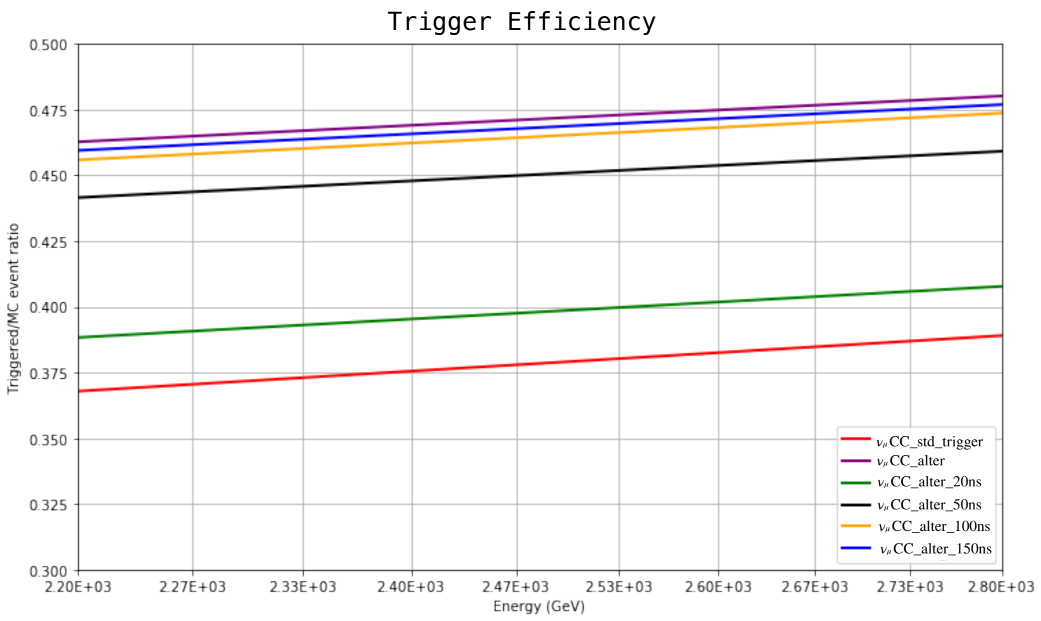


FIGURE 2.8 Trigger efficiency (vs. neutrino energy) dependency on time window - zoom.

Atmospheric muons background rate

All of the previously discussed trigger conditions have been tested also for the detection of atmospheric muons. Being the latter a source of background, their signal rate must be kept low, and a tolerable value is around 100 Hz (as stated above). Table 2.5 shows the rate for 1 million simulated events with a threshold at 100 GeV for one block of

Trigger configuration	**Background rate (Hz)**
std	61
50 ns	72
100 ns	80
150 ns	83
alter	84
st_nocomb	$1.3 \cdot 10^2$

TABLE 2.5 Table of comparison of the atmospheric muons background rate for different trigger conditions.

ARCA and livetime of 2530 seconds. The conditions ensuring the minimum trigger rate are the standard trigger one, while the deactivation of the parameter `combineL1` doubles that rate. As for the other trigger conditions, the background rate for the atmospheric muons is acceptable.

2.3.3 Standard trigger vs. *(*100***ns,*** 4 ***hits)*** on reconstructed events

Since the amount of events lost at trigger level (i.e. the number of events after the trigger compared to the initial number of simulated events) is almost half of the total number of events, a further investigation on trigger conditions allowing more events to be triggered is worthwhile. The choices of parameter sets investigated in this section are the following tuples: $(20ns, 5hits)$, also known as the standard trigger conditions (**std trigger**), and $(100ns, 4hits)$, which triggers more events, keeping the background rate acceptable. For the trigger parameters optimisation, though, the trigger efficiency is not the only quantity to be maximised. In fact, a good selection of events should lead to a good reconstruction. The triggered events are then processed with the reconstruction algorithm provided by `JPP`, to estimate the muon energy and direction, through the following chain: `JPrefit, JSimplex, JGandalf, JEnergy`. In details, `JPrefit` selects clusters of L0 and L1 hits by running a scan of directions in the solid angle; for each direction, a three parameter (x, y, t) fit is performed on the data; the output of `JPrefit` is processed with `JSimplex`, in which a five parameter (x, y, t, dx, dy) fit is applied to the data (where dx and dy are the x and y neutrino direction components). From the previous step, the best 12 fits obtained with `JPrefit` are compared and the one yielding the smallest angular discrepancy between the reconstructed track and the simulated direction is selected. The ordering of the solutions follows a reconstruction quality parameter, related to the number of hits, the number of degrees of freedom and the χ^2 error for the fit. From the `JSimplex` step, only one solution is kept and considered as the best fit. The `JGandalf` fit is based on the Levenberg-Marquardt optimisation algorithm [33, 34]. In this fit, all hits within a preset road width and interval around the expected arrival time are selected. Quality parameters such as the likelihood (of the fit w.r.t. the original direction) and the χ^2 of the fit are calculated. Eventually, `JEnergy` estimates, from the best of the previous fits, the energy of the neutrino.

Reconstruction outputs, containing fit parameters, the position and the direction of the lepton produced by the neutrino interaction, an estimation of the muon neutrino energy, as well as likelihood and quality parameters, are used as a further check to assess if the choice of the trigger conditions is optimal. The leptons considered in this analysis are muons, since this reconstruction algorithm relies on a track hypothesis. The reconstructed direction is used to calculate the angle between the simulated and reconstructed track, and so to define *well reconstructed events.* The angle α is calculated from the direction cosines of the two tracks, as the arc-cosine of the scalar product: $\alpha = acos(mc\_\vec{dir} \cdot reco\_\vec{dir})$. A well reconstructed event is defined after a threshold value for the angle α is chosen. Thus, for a given choice of α, the number of reconstructed events is compared to the total simulated events in each energy bin. In order to evaluate the dependency of the number of well reconstructed events on the angle between simulated and reconstructed tracks, for the two considered choices of trigger parameters the fraction of well reconstructed over the total number of simulated events has been evaluated, setting a threshold to separate low and high energies at 10 TeV. Tables 2.6 and 2.7 show the results for the two sets of parameters, separately, without any source of background.

Angle α (deg)	**% well reconstructed events ($E \leq 10$ TeV)**	**% of well reconstructed events ($E > 10$ TeV)**
0.0	0	0
0.2	3.4	29.7
0.4	8.0	39.5
0.6	11.8	42.5
0.8	12.7	44.0
1.0	13.9	45.0
1.2	14.8	45.2
1.4	15.4	45.6
1.6	15.9	45.9
1.8	16.2	46.2
2.0	16.6	46.4

TABLE 2.6 Fraction of well reconstructed events at different allowance angles, for low and high energy events separately, for **std trigger** conditions.

The reconstruction efficiency is defined as the ratio between *well reconstructed* and simulated events per bin. The efficiency, evaluated for each set of trigger conditions, is compared in the plot shown in Figure 2.9 for different allowance angles.

Angle α (deg)	% of well reconstructed events ($E \leq 10$ TeV)	% of well reconstructed events ($E > 10$ TeV)
0.0	0	0
0.2	3.7	30.1
0.4	8.7	40.1
0.6	12.0	43.4
0.8	14.2	45.0
1.0	15.7	45.8
1.2	16.8	46.4
1.4	17.5	46.8
1.6	18.1	47.2
1.8	18.6	47.4
2.0	19.0	47.7

TABLE 2.7 Fraction of well reconstructed events at different allowance angles, for low and high energy events separately, for $(100ns, 4hits)$ conditions.

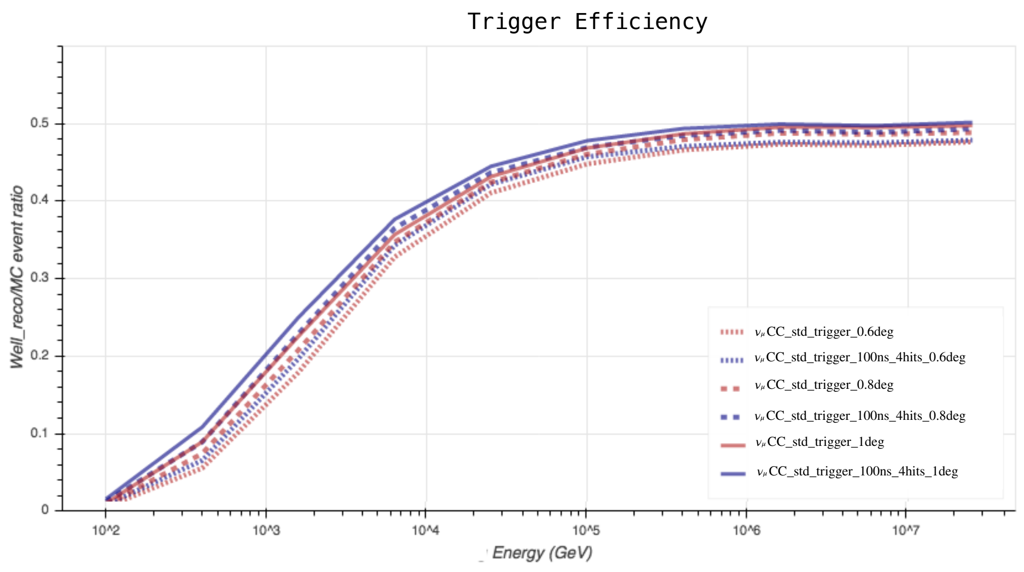


FIGURE 2.9 Reconstruction efficiency for different allowance angles.

As reported in Table 2.6 and Table 2.7, as well as in Figure 2.9, the reconstruction efficiency is always below 0.5, meaning that at least half of the events is lost in the process. Moreover, the $(100ns, 4hits)$ set of conditions allows more events to be triggered and then reconstructed, especially in the low energy region.

Reconstructed events with random background

To perform a more realistic study of the reconstruction efficiency, the simulated events have also been processed adding random background using the *background* parameter (i.e. `B`) in `JTriggerEfficiency`. This option creates random hits in the detector volume, to add noise with a chosen frequency to the detected signal. As a result, more hits are triggered, even though they are not correlated to the original simulated event. The reconstruction efficiency is then evaluated using the events processed with the random background (with the default input rate, namely `"5000, 500, 50, 5, 0.5"` (Hz)), and the results are shown in Figure 2.10. Again, $(100ns, 4hits)$ gives a higher efficiency, especially in the low energy region: with the $(100ns, 4hits)$ requirements more events are collected w.r.t. the $(20ns, 5hits)$ case. On the other hand, the efficiency curves with and without background are almost coincident in the case of the standard trigger.

As a further check, the ratio between the reconstruction efficiency, for both choices of parameters, for the background case w.r.t. the "background-free" one has been evaluated and is shown in Figure 2.11. This can be interpreted as the standard trigger selection giving many more pure events as a result, at least in the low energy region.

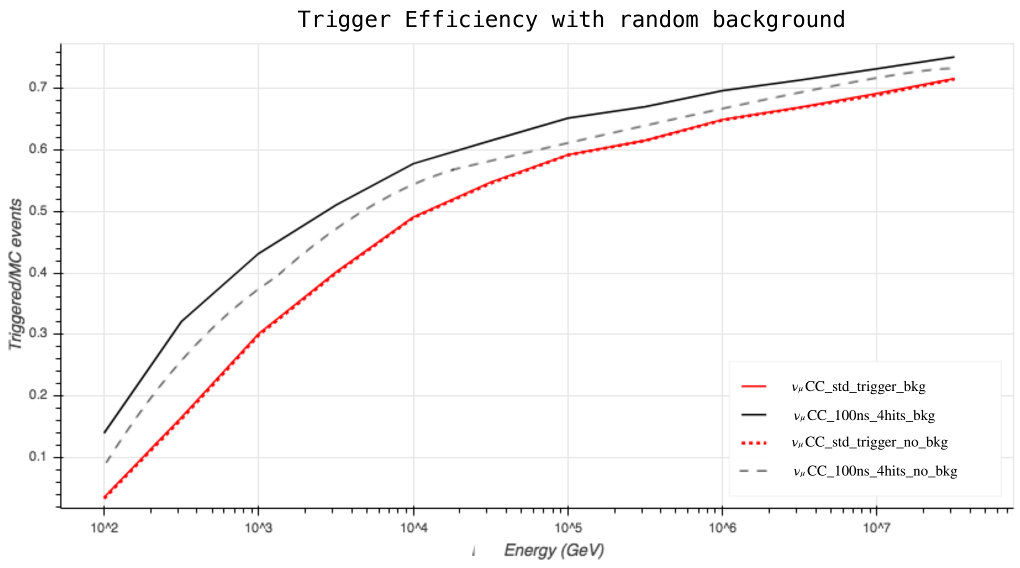


FIGURE 2.10 Trigger efficiency as a function of the neutrino energy with and without the random background parameter activated in JTE for **std trigger** and $(100ns, 4hits)$.

To summarise, it has been seen in the previous sections that the time window plays a crucial role in the selection of events, as well as the minimum number of hits. Moreover, selecting the number of optical modules hit in the given timeslice allows fewer random coincidences to be selected and thus provides event samples with higher purity. Concerning the background from atmospheric muons, both trigger configurations tested give a reasonable amount of random background detected, even though the rate is slightly higher for the one with the larger time window. Despite the fact that almost half of the simulated events are being lost with any of the trigger conditions tested, as shown in Figure 2.9, it is seen that a larger amount of random coincidences

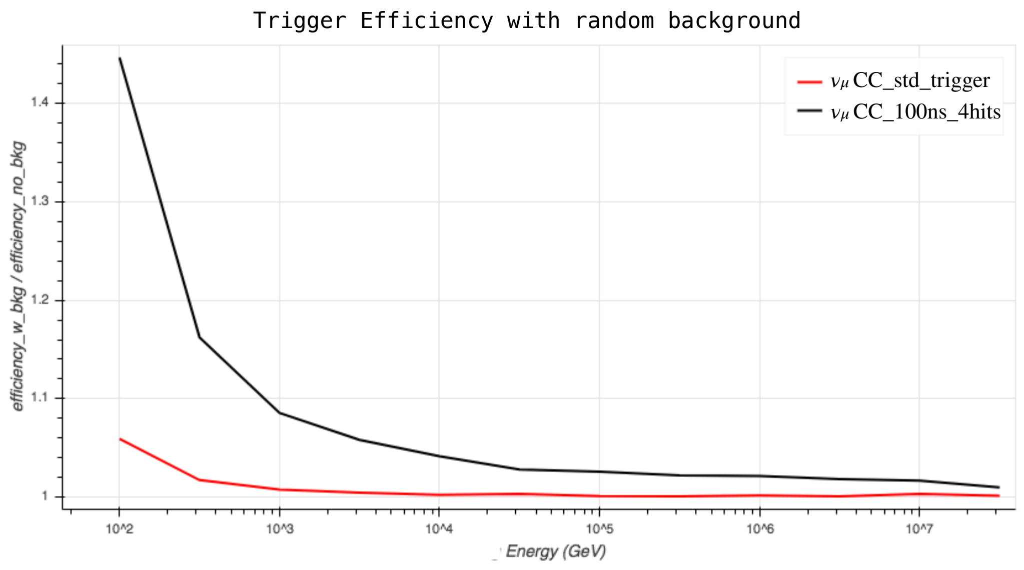


FIGURE 2.11 Ratio of trigger efficiencies with and without random background for **std trigger** and $(100ns, 4hits)$.

is collected with the $(100ns, 4hits)$ choice, especially in the low energy region, which is also dominated by other sources of background.

As a consequence, in order to select more pure events, which are consequently well reconstructed, it is convenient to use **std trigger** conditions, which also keep a reasonable background rate. This choice is safe in this phase in which the trigger and reconstruction algorithms are still being studied and optimised on both simulated and real-data events, and is especially convenient for the analyses shown in the following sections, as the trigger represents the starting point for the Neural Network analysis reported in Chapter 4. In fact, the events used as input are the triggered events (using **std trigger** conditions) with at least 5 hit DOMs. An inefficient choice of the trigger conditions, resulting in wrongly selected events, would affect the Neural Network study and the derived conclusions.

Chapter 3

Machine and Deep Learning

Machine Learning is defined as "the systematic study of algorithms and systems that improve their knowledge or performance with experience." [35]. In fact, with the term Machine Learning (also ML from here on) we refer to the category of algorithms and statistical methods that are able to automatically extract *meaningful information* from (typically large) data sets [36]. *Data-Driven* methodologies are adopted by Machine Learning techniques to learn to *generalise*: input data are used to tune the behaviour of the algorithm, so that it will be possible to generate predictions on similar but unseen data. Such generalisation capability practically defines the concept of *learning through experience* that is the specific characteristic differentiating ML approaches from classical "data mining" algorithms.

Tom Mitchell in [37] defines Machine Learning as:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E"

On the other hand, Peter Flach in [35] defines Machine Learning from a slightly different perspective:

> "Machine learning is concerned with using the right *features* to build the right *models* that achieve the right *tasks*.".

This definition emphasises the fundamental "ingredients" (as the author calls them) constituting ML techniques.

The science of learning plays a key role in the fields of statistics, data mining and artificial intelligence, intersecting with areas of engineering and other disciplines [38]. However, differently from the "classical" approach to Artificial Intelligence (AI), Machine Learning introduces a shift in the adopted programming paradigm [39]. In classical AI, the programming paradigm was based on symbolic computation: humans input rules (i.e. a program), and data are then processed according to these rules, so that answers can be generated (see Figure 3.1).

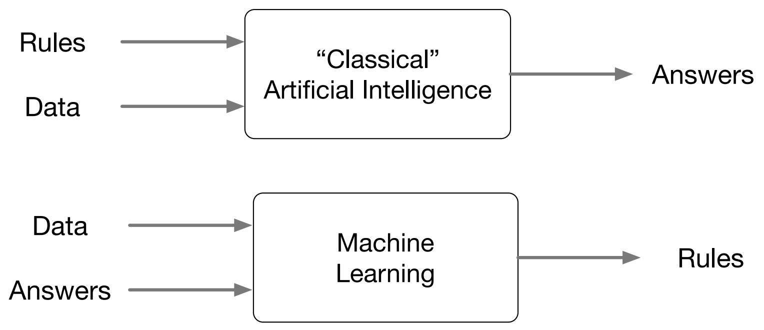


FIGURE 3.1 Machine Learning: a different programming paradigm [39].

With Machine Learning, humans input data as well as the answers expected from the data, so that the rules generating the answers can be derived. These rules can then be applied to new data to produce original answers. In fact, Machine Learning is about learning from data [38]. However, differently from classical AI, Machine Learning systems are *trained* rather than explicitly programmed. In a typical scenario, there is an outcome measurement, usually *quantitative* (such as a stock price) or *categorical* (such as *heart attack/no heart attack*), that has to be estimated based on a set of features (e.g. diet and clinical measurements). A *training set* of data is available, in which the outcome and feature measurements for a set of objects (e.g. people) can be found. Using these data, it is possible to build a prediction model, or *learner*, which will enable to estimate the outcome for new unseen objects. A good learner is one that accurately estimates outcomes for unseen data.

In more details, developing ML applications requires the following steps to be accomplished [36]:

1. *Problem formulation*: The first crucial step concerns the formulation of a given problem in terms of the selected ML approach. There are several learning methods that are based on different theoretical backgrounds and adopt different algorithmic strategies. All these aspects characterise the *task* of different ML techniques, which must be taken into account during the problem formulation.

2. *Problem representation*: The next step is to select an appropriate representation for both the data and the knowledge to be learned (i.e. the *model*). Different learning methods require different formalisms, e.g. some approaches are based

on a vectorial representation of data, while others require to process structured data such as trees or graphs.

3. *Data collection*: Data represent the main "ingredient" for ML algorithms. In particular, ML approaches strongly depend on the quality and the quantity of the data available to carry out the learning process. Therefore, data must be properly pre-processed to remove spurious values and fitting a unique format. Several techniques of *data processing* are required to tackle this problem.
   Furthermore, data are fundamental for the definition of *features*, which determine much of the success of ML applications, because a model is only good as its features [35]. A feature can be thought of as a function that maps the data from one domain to another, i.e. the *feature space*.

4. *Perform the learning process*: Once the data have been collected, the learning process can be accomplished. This step represents the core of ML algorithms [36], where data are actually analysed. Depending on the specific strategy, data may be separated in two different sets, namely the *training* and the *test* sets, exploited in an iterative process to train and evaluate the learning, respectively.

5. *Analyse and evaluate the learnt knowledge*: The final step of ML applications is devoted to the evaluation of the performances of the learning process. Depending on the type of application, this step could also be an integral part of the learning process itself. Besides performance evaluations, this step is also important to discover and resolve practical problems that may affect the learning process itself, such as *overfitting*, or the *local optima* [40]. Some possible causes could be data inadequacy, noise or irrelevant attributes in data.

Figure 3.2 summarises these steps, as reported in [35].

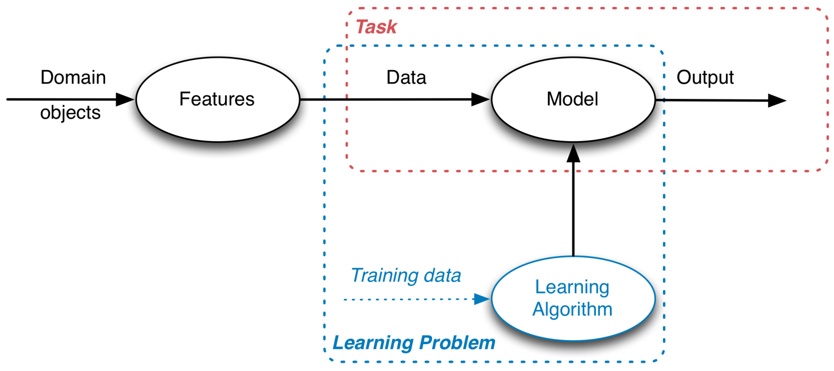


FIGURE 3.2 Machine Learning algorithm representation [35].

Machine Learning lies at the intersection of computer science, engineering, and statistics [36], and it can be applied to many problems. In fact, ML algorithms have proven to be of great value in a variety of application domains: any field that needs to interpret and act on data can benefit from Machine Learning techniques [36].

Deep Learning (also DL from here on) is a subfield of machine learning, which puts an emphasis on learning successive *layers* of increasingly meaningful representations [39]. In fact, the term *deep* stands for the idea of multiple stacked layers, and the number of layers defines the *depth* of the model. Other appropriate names for the field could have been *layered representations learning*, or *hierarchical representations learning* [39]. Modern deep learning often involves tens or even hundreds of successive layers of representations and they are all learned automatically from training data. Conversely, other Machine Learning approaches tend to focus on learning only one or two layers of representations of the data, so they are sometimes referred to as *shallow learning*. In Deep Learning, such layered representations are (most commonly) learnt via models called *Neural Networks*, structured in layers stacked on top of each other.

Data Augmentation

One of the best ways to make Machine Learning models generalise better is to train them on large sets of data. This is particularly the case of Deep Learning Neural Networks, in which the quality of the feature representations learnt from the data through the multiple stacked layers improves with the increasing size of the dataset. However, as most datasets in practice are limited, this is not always possible.

Data augmentation is a technique generally used to improve the training phase of Machine Learning models by providing slightly modified versions of the same examples. The typical application of these techniques is on images, in which operations on data are easily applicable, usually involving in-plane rotations and symmetric transformations. Including these artificially transformed samples allows Machine Learning models to be more robust to noisy input data. In fact, providing different versions of the same samples forces the model to learn so-called *translation-invariant* (resp. *rotation-invariant*) representation of features.

3.1 Machine Learning Settings

Although ML systems may be classified according to different points of view [41], a common choice is to classify ML approaches according to the specific learning strategy, namely the strategy ML approaches adopt to learn from data.

In every learning situation, the learner transforms information provided by the environment into some new form in which it is stored for future use [37, 42]. The nature of this transformation determines the type of learning strategy adopted. Several basic strategies have been distinguished: *rote learning*, *learning by instruction*, *learning by deduction*, *learning by analogy*, and *learning by induction* [42]. The latter is further distinguished in *learning by observation* and *learning from examples* [41, 42].

Learning from examples is one of the most popular and widely employed strategy in ML approaches that is often simply called *learning.* Similarly, the term “example” is usually treated as a synonym for “data”. Thus, from now on, the two terms will be used interchangeably.

The *learning problem* that this strategy involves can be described as “finding a general rule that explains the data, given only a sample of limited size”. This strategy comprises multiple learning techniques, further organised in three main categories, i.e. *Supervised Learning*, *Reinforcement Learning*, and *Unsupervised Learning* [43].

3.1.1 Supervised Learning

In Supervised Learning, data are represented as tuples in the form of $\langle input, output \rangle$ patterns. This strategy is called *supervised* because the objects under considerations are already associated with the target values, i.e. the *labels.* In more details, in the problem of Supervised Learning, the training set of examples is provided, along with corresponding correct outcomes. Based on this training set, the learning algorithm generalises to respond correctly to all possible inputs [43]. More formally, the training set is usually written as a set of pairs (x_i, t_i), where the inputs are x_i, the targets are t_i, and the i index suggests that there are lots of pairs, ranging from 1 to an upper limit $N \in \mathbb{N}$ [43]. Each index i refers to a specific training *sample.*

Typical examples of Supervised Learning approaches are the *Classification* and the *Regression* that differ according to the type of the outcomes (and so, the labels):

- **Classification**: In the *classification* learning problem, the output space is composed by a finite number of discrete *classes*, and the corresponding learning algorithm is called *classifier.* In particular, the learning problem is to assign to each input data its corresponding class.

- **Regression**: If the output space of the learning problem corresponds to values of continuous variables, then the learning task is known as the problem of *regression.* Typical examples of regression include predicting the value of shares in the stock exchange market, or estimating the value of a physical quantity like the direction of a particle, or its energy.

In many cases, the outputs y may be difficult to collect automatically and must be provided by a human "supervisor", hence the supervised label, but the term applies also to cases in which the training set targets are collected automatically [43].

3.1.2 Reinforcement Learning

Reinforcement learning corresponds to a strategy that lies between Supervised and Unsupervised learning. The ML algorithm gets told when the output is wrong, but does not know how to correct it [43]. Thus, the algorithm tries to explore different possibilities until the final correct output has been discovered. In other words, the problem of reinforcement learning is to learn what to do, i.e. how to map situations to actions in order to maximise a given *reward.* Such maximisation is what guides the learning algorithm through the different possible solutions. [44]

3.1.3 Unsupervised Learning

If the input data to the learning algorithm comprises a set of samples without associated labels, the problem is classified as *Unsupervised Learning.* In Unsupervised Learning, data do not contain any indication to the correct target. Instead, the algorithm tries to identify *similarities* among the inputs so that inputs that are in some way related are *categorised* together [43]. Some of the advantages of Unsupervised Learning techniques with respect to Supervised ones are [45]:

- There is no cost of collecting and labelling samples.
- Unsupervised techniques may be used to identify characteristics of the samples which are useful for differentiating among them.
- Unsupervised techniques may be used for exploring the data analysing their structure.

In the rest of the Chapter we will limit the discussion to Supervised Learning settings, as this is the main focus of the presented work. A more complete reference to Unsupervised Learning settings can be found in [35, 38, 40, 46].

3.2 Deep Learning

As briefly mentioned before, Deep Learning is a specific subfield of Machine Learning in which the main focus is on learning from data, through successive layers of increasingly meaningful representations. In Deep Learning, these layered representations in data are learnt by using a specific set of Machine Learning models named *artificial neural networks* (ANN). In other words, Deep Learning refers to the set of algorithms and methods developed to train ANN with many layers most efficiently [47].

The term *neural network*[1] is a reference to neurobiology, but although some of the central concepts in Deep Learning were developed in part by drawing inspiration from our understanding of the brain, Deep Learning models are not models of the brain [39]. Hence, for our purposes, Deep Learning only represents a mathematical framework that will be used to learn (meaningful) representations from data.

3.2.1 Modelling complex functions with Artificial Neural Networks

Despite the huge interest gained in recent years on Deep Learning and Neural Networks, early studies go back to the 1940s when Warren McCulloch and Walter Pitt proposed the first scheme of how neurons could work [48]. However, in the decades that followed the first implementation of the McCulloch-Pitt neuron model, the so-called *Perceptron* [49], many researchers and Machine Learning practitioners slowly began to lose interest in Neural Networks since nobody had a good solution for training a neural network with multiple layers [47]. The interest in Neural Network was revived only in 1986, when G.E. Hinton *et al.* (re)discovered and popularised the *back-propagation* algorithm to train ANN more efficiently [50].

Nevertheless, the *Perceptron*, that is the first model of ANN with only one single layer, constitutes the building block on which multi-layer ANN are defined [47].

[1]Note that the terms *artificial neural networks*, *neural networks*, and ANN are all synonyms, and so will be used interchangeably.

Single-layer Neural Network

The processing unit of the human brain is called *neuron.* There are lots of them ($\approx 10^{11}$ according to the most common estimation [43]), and each neuron comes in lots of different types, depending on its particular task. However, their general operation is similar in all cases: transmitter chemicals within the fluid of the brain raise or lower the electrical potential inside the neuron. If this *membrane potential* reaches some threshold, the neuron *spikes* (or *fires*), and a pulse of fixed strength and duration is sent down the *axon* [43]. The axons organise neurons into specific series of interconnections named *synapses.*

This simplistic schematisation of the human brain structure and activity is what McCulloch and Pitts in [48] tried to extract and represent in their mathematical model. In more details, they proposed a model of neurons consisting of [43]:

- *a set of weighted inputs* w_j, that correspond to the synapses;
- an *adder* that sums the input signals (equivalent to the membrane of the cell that collects electrical charge)
- *an activation function* (initially a *threshold function*) that decides whether the neuron fires for the current inputs.

A representation of this model is reported in Figure 3.3. This scheme will be used as reference to report the corresponding mathematical description.

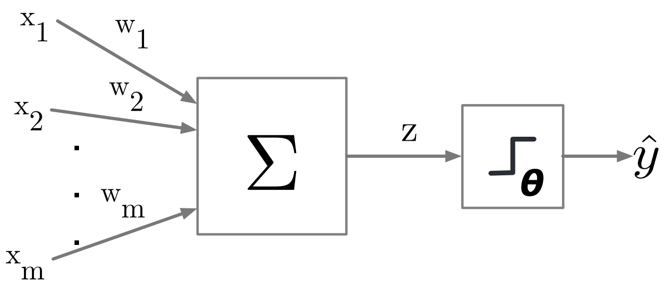


FIGURE 3.3 A picture of McCulloch and Pitt's mathematical model of a neuron. The input features x_j are multiplied by the weights w_j, and the neurons sum their values. If this sum is greater than the threshold θ, then the neuron fires, otherwise it does not.

On the left-hand side of the Figure, there is the set of the input nodes, namely $(x_1, x_2, \ldots, x_m)$. The strength of the synapsis is represented by the weights w_j, associated to each connection. This strength affects the strength of the signal, so each input is multiplied by the associated weight, i.e. $x_j \times w_j$. When all the signals arrive into the neuron, it adds them up to see if there is enough strength to make it fire, namely:

$$z = \sum_{j=1}^{m} w_j x_j = \mathbf{w}^T x, \tag{3.1}$$

The McCulloch and Pitts neuron is a binary threshold model: it sums up the inputs (multiplied by the synaptic weights), and either the neuron fires or does not fire. Thus, if $z > \theta$, the neuron will produce output $\hat{y} = 1$ (i.e. fires); $\hat{y} = 0$, otherwise. So, the *activation* function of the network can be formalised as:

$$\hat{y} = \phi(z) = \begin{cases} 1 & \text{if } z > \theta \\ 0 & \text{if } z \leq \theta \end{cases} \tag{3.2}$$

This simple *unit step function*, is also known as the *Heaviside step function.* However, to simplify the notation, we can bring the threshold parameter θ to the left side of the equation, and reformulate z as a weight-zero linear combination:

$$z = \sum_{j=0}^{m} w_j x_j \tag{3.3}$$

where $w_0 = -\theta$ and $x_0 = 1$. Consequently:

$$\hat{y} = \phi(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases} \tag{3.4}$$

In the literature, this is called the *bias input trick* [43].

The Perceptron model proposed by Rosenblatt in [51], is a collection of McCulloch and Pitts neurons together with a set of inputs and some weights to fasten the inputs to the neurons [43]. The main advantage introduced by this model is that it consists of an algorithm able to learn automatically the optimal weight coefficients w_j. In the context of Supervised Learning, and classification, this algorithm could then be used to predict if a sample belongs or not to one class. In the literature, this problem is usually referred to as *binary classification problem.*

The whole idea behind the Perceptron model is fairly simple. Considering $\mathbb{X} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(i)}, \ldots, \mathbf{x}^{(N)}\}$ as the input (training) data to the model, so that

$$\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \ldots, x_j^{(i)}, \ldots, x_m^{(i)}) \in \mathbb{X} \tag{3.5}$$

and $\mathbf{w} = (w_1, w_2, \dots, w_j, \dots, w_m)$ the weight vector, the Perceptron algorithm can be summarised by the following steps [47]:

1. Initialise the weights $\mathbf{w}$ to $\mathbf{0}$ or vector of small random numbers;
2. For each training sample $\mathbf{x}^{(i)}$ perform the following steps:
   (a) Compute the output value $\hat{y} = \phi(z)$;
   (b) Update the weights.

The simultaneous update of each weight w_j in the weight vector $\mathbf{w}$ can be formally defined as:

$$w_j = w_j + \Delta w_j \tag{3.6}$$

The value of Δw_j, which is used to update the weight w_j, is calculated by the so-called *perceptron learning rule* [43, 47]:

$$\Delta w_j = \eta(\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)})\mathbf{x_j}^{(i)} \tag{3.7}$$

where η is the *learning rate*, $\mathbf{y}^{(i)}$ is the true label of the i-th training sample, $\hat{\mathbf{y}}^{(i)}$ is the corresponding predicted label, and $E = (\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)})$ is the error function, namely a function to calculate the number of misclassifications obtained by the model. It is important to remark that all the weights in $\mathbf{w}$ are being updated simultaneously, which means that $\hat{\mathbf{y}}^{(i)}$ is not recomputed before all of the weights Δw_j were updated.

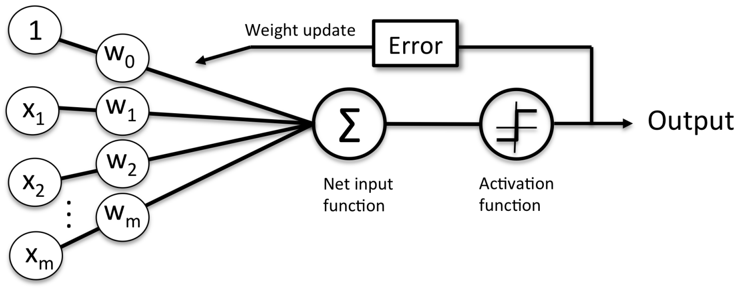


FIGURE 3.4 The Perceptron learning strategy [47].

Figure 3.4 illustrates the learning strategy as defined by the Perceptron model. The convergence of the Perceptron model is only guaranteed if the considered binary classification problems consists of two classes that are *linearly separable*, and the learning rate η is sufficiently small [35]. This is a direct consequence of the defined learning function that is based on a linear model by definition. Alternatively, it is possible to set a maximum number of steps over the training dataset, called *epochs*, and/or a threshold for the number of tolerated misclassifications, to prevent the algorithm from updating the weights indefinitely.

The Adaline model (**ADA**ptive **LI**near **NE**uron [52] can be considered an evolution of the Perceptron model. This algorithm is particularly interesting because it illustrates the key concept of defining and minimising cost functions, which represent the very basis to understand how the learning of a Neural Network works. The key difference between the Adaline model and the Perceptron is that the weights w_j are updated based on a linear activation function rather than a unit step function. The Adaline learning strategy is represented in Figure 3.5. In this model, the linear activation function $\phi(z)$ is simply the identity function of the net input so that $\phi(\mathbf{w}^T\mathbf{x}) = \mathbf{w}^T\mathbf{x}$.

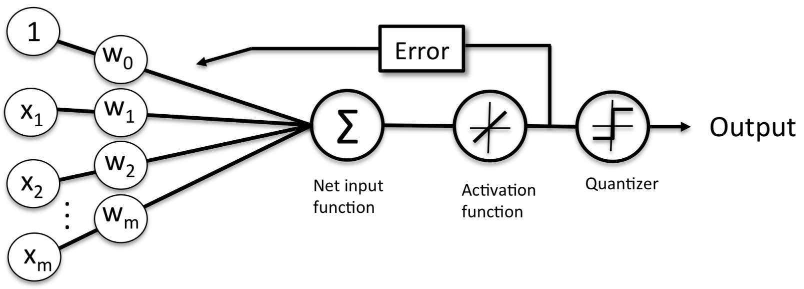


FIGURE 3.5 The Adaline learning strategy [47].

Differently from the learning strategy illustrated in Figure 3.4, the difference in the Adaline strategy is to use the continuous valued output from the linear activation to compute the model error and update the weights accordingly, rather than binary class labels.

One of the key ingredients of Supervised Learning algorithms is to define an *objective function* that has to be optimised during the learning process. This objective function is often a *cost function* (similarly *loss function*) that we want to minimise [47]. In the case of Adaline, the cost function J to minimise to learn the weights is the *sum of squared errors* (SSE) between the calculated outcome and the true class labels:

$$J(w) = \frac{1}{2}\sum_i (\mathbf{y}^{(i)} - \phi(\mathbf{z}^{(i)}))^2 \tag{3.8}$$

The main advantage of this continuous linear activation function is that the cost function is differentiable (in contrast to the unit step function used by the Perceptron). Furthermore, this cost function is convex; thus, we can use a simple, yet powerful, optimisation algorithm called *gradient descent* [40] to find the weights that minimise the cost function (see Figure 3.6).

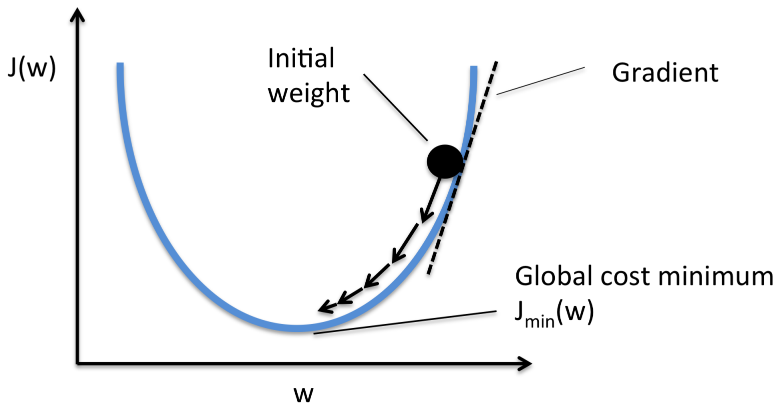


FIGURE 3.6 The Gradient Descent optimisation algorithm [47].

Using the gradient descent, the weight update rule can be expressed as taking a step away from the gradient $\nabla J(\mathbf{w})$ of the cost function $J(\mathbf{w})$:

$$\mathbf{w} := \mathbf{w} + \Delta\mathbf{w} \tag{3.9}$$

In this formulation, the weight change $\Delta\mathbf{w}$ is defined as the negative gradient[2] multiplied by the learning rate η:

$$\Delta\mathbf{w} = -\eta\Delta J(\mathbf{w}) \tag{3.10}$$

so that

$$\Delta w_j = -\eta\frac{\partial J}{\partial w_j} = -\eta\sum_i (\mathbf{y}^{(i)} - \phi(\mathbf{z}^{(i)}))x_j^{(i)} \tag{3.11}$$

Although the Adaline learning rule looks similar to the Perceptron rule, the weight update is calculated based on all samples in the training set, instead of updating the weights incrementally after each sample. This approach is also referred to as (*full*) *batch* gradient descent [47]. However, in real Deep Learning settings, in which the full batch gradient descent strategy can be computationally expensive, the *stochastic gradient descent* algorithm (SGD) is used instead [40].

[2]As we are interested in minimising the cost function

The Multilayer Perceptron

The Multilayer perceptron (MLP) [49, 53], (also referred to in the literature as the *traditional* Neural Network model), has been defined as an evolution of the single-layer Neural Network. Multi Layer Perceptron can be considered as the joining link between traditional ML and DL, as its simplest structure consists of one (passthrough) input layer, one *hidden* layer, and an output layer (see Figure 3.7). Although the name may suggest differently, the original definition of the MLP model consists of only one hidden layer. In fact, according to Hinton [54], a *deep network* differentiates from the MLP for having at least more than one hidden layer.

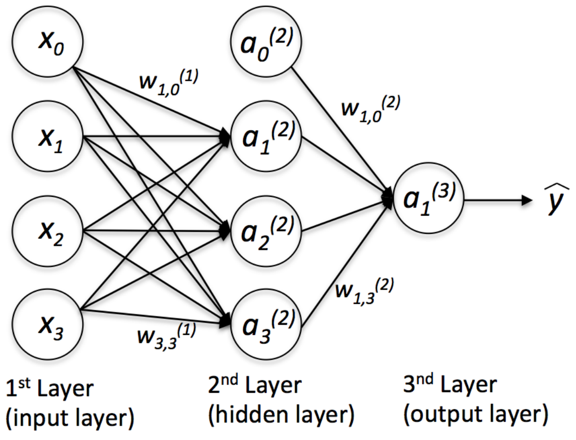


FIGURE 3.7 The Multilayer Perceptron model [47].

All the three layers of MLP are *fully connected*[3], meaning that the output of each node is the weighted sum of the outputs of all nodes in the previous layer, plus a bias term, operated on by a non-linear function. Traditionally, the preferred non-linearity is a sigmoid function such as tanh or the logistic function [53]. A more comprehensive description of Neural Network activation functions is reported in Section 3.3.

A MLP with a single hidden layer, under certain assumptions, can be shown to approximate any function to arbitrary precision given a sufficient number of hidden nodes [55]. The MLP is a powerful technique, but it has a number of deficiencies [56]. For example, it tends to scale poorly to a large number of input samples. Nevertheless, the MLP model represents the fundamental basis on which modern Deep Learning architectures have been defined. The learning process of the MLP will be used as reference to describe the learning mechanism adopted by Neural Networks to learn from input data (see Section 3.3).

[3]Also referred to as *dense* layers

3.3 Training Artificial Neural Networks

The training process of an ANN is rather simple: input data are passed to the network, and the output of every neuron in each consecutive layer is computed. This phase of the process is called the *forward* pass, as data flows from input layer to output layer. Once the forward pass is completed, the network output error, namely the difference between the expected output and the generated estimations, is computed. This calculation then proceeds *backward*, i.e. from output layer up to the input layer: first it computes how much each neuron in the last hidden layer contributed to each output neuron's error; then it proceeds to measure how much of these error contributions came from each neuron in the previous hidden layer and so on, until the algorithm reaches the input layer. This reverse pass efficiently measures the error across all the connection weights in the network, by propagating the error backward in the network (hence the name of the algorithm, i.e. *backward propagation*). Within this process, a lot of components of the network are involved, such as layers' activation functions, or the optimisation technique used to calculate the gradients. Some of these components will be detailed in the rest of this Section. A more comprehensive reference can be found in [57].

3.3.1 Neural Network Training

The learning procedure of ANN consists of three simple steps:

1. Starting at the input layer, input data are forward propagated through the network to generate the output (i.e. predictions);
2. Based on generated predictions, the error to minimise is calculated, using the specified loss function;
3. The error is back-propagated in the reverse order (from the output layer to the input one), so that the derivatives with respect to each weights are calculated, and the model parameters are updated.

The above steps are repeated several times (i.e. *epochs*). In the following, a more formal definition of the *forward* and *backward* pass will be provided. Without loss of generality, the description will be limited to the case of MLP (i.e. one single hidden layer).

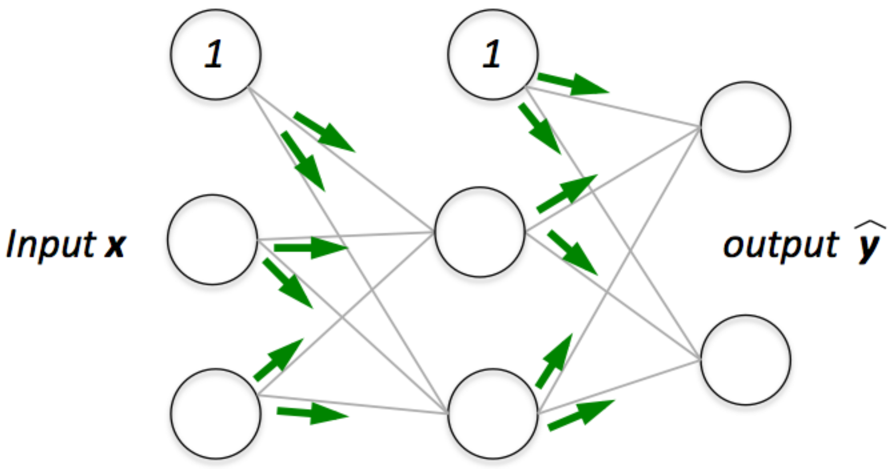


FIGURE 3.8 The forward propagation step of MLP [47].

Forward Propagation

Since each unit in the hidden layer is connected to all the units in the input layer, the activations $\mathbf{a}^{(2)} = \phi(\mathbf{z}^{(2)})$, where

$$\mathbf{z}^{(2)} = \mathbf{W}^{(1)}\mathbf{a}^{(1)} \tag{3.12}$$

Here, $\mathbf{a}^{(1)}$ is a $(m+1) \times 1$ dimensional feature vector of a sample $\mathbf{x}^{(i)}$ plus bias unit. $\mathbf{W}^{(1)}$ is a $h \times (m+1)$ dimensional weight matrix; h is the number of hidden units. $\mathbf{z}^{(2)}$ is the $h \times 1$ dimensional net input vector obtained after the matrix-vector multiplication. Finally, $\mathbf{a}^{(2)} \in \mathbb{R}^{h\times 1}$ is calculated.

The generalisation of the formulation for all the N samples in the training set can be written as:

$$\mathbf{Z}^{(2)} = \mathbf{W}^{(1)}[\mathbf{A}^{(1)}]^T \tag{3.13}$$

Here, $\mathbf{A}^{(1)}$ is the $N \times (m+1)$ matrix, and the matrix-matrix multiplication will result in a $h \times N$ dimensional input matrix $\mathbf{Z}^{(2)}$.

Similarly, the same procedure is repeated to calculate $\mathbf{A}^{(3)}$, i.e. the activation of the output layer in the vectorised form:

$$\mathbf{Z}^{(3)} = \mathbf{W}^{(2)}[\mathbf{A}^{(2)}] \tag{3.14}$$

$$\mathbf{A}^{(3)} = \phi(\mathbf{Z}^{(3)}), \mathbf{A}^{(3)} \in \mathbb{R}^{t\times N} \tag{3.15}$$

where t is the number of output units. A schematic representation of the forward propagation step is reported in Figure 3.8.

Backward Propagation

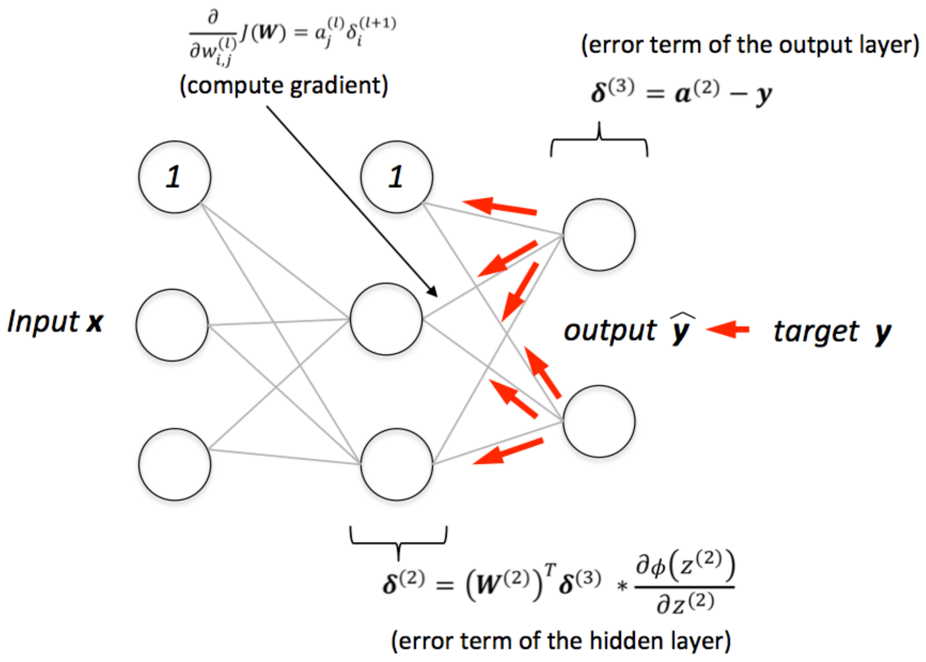

$\frac{\partial}{\partial w_{i,j}^{(l)}} J(\boldsymbol{W}) = a_j^{(l)} \delta_i^{(l+1)}$
(compute gradient)
(error term of the output layer)
$\boldsymbol{\delta}^{(3)} = \boldsymbol{a}^{(2)} - \boldsymbol{y}$
1
1
Input x
output $\hat{\boldsymbol{y}}$
target y
$\boldsymbol{\delta}^{(2)} = (\boldsymbol{W}^{(2)})^T \boldsymbol{\delta}^{(3)} * \frac{\partial \phi(z^{(2)})}{\partial z^{(2)}}$
(error term of the hidden layer)

FIGURE 3.9 The backward propagation step of MLP [47].

In the back-propagation algorithm, the objective is to propagate the error from right to left. Thus, the error vector of the output layer must be calculated first, which in the case of the MLP corresponds to:

$$\delta^{(3)} = \mathbf{a}^{(2)} - y \tag{3.16}$$

Next, the error term of the hidden layer is calculated:

$$\delta^{(2)} = (\mathbf{W}^{(2)})^T \delta^{(3)} \frac{\partial \phi(\mathbf{z}^{(2)})}{\partial \mathbf{z}^{(2)}} \tag{3.17}$$

where $\frac{\partial \phi(\mathbf{z}^{(2)})}{\partial \mathbf{z}^{(2)}}$ is the derivative of the activation function of the hidden layer (usually a logistic function in the case of MLP). Now, generalising this calculation to every sample in the training set, $\boldsymbol{\Delta}^{(l)}$ can be defined as:

$$\boldsymbol{\Delta}^{(l)} := \boldsymbol{\Delta}^{(l)} + \delta^{(l+1)}(\mathbf{A}^{(l)})^T \tag{3.18}$$

Last, after the gradients have been computed, the weights can be updated by taking a step opposite to the gradient:

$$\mathbf{W}^{(l)} := \mathbf{W}^{(l)} - \eta\mathbf{\Delta}^{(l)} \tag{3.19}$$

The back-propagation steps are illustrated in Figure 3.9.

3.3.2 Activation Functions

One of the key advantages of the back-propagation is that the algorithm can be used whatever activation function is chosen within layers, with the only constraint imposed by the method that the activation function must be differentiable.

As in the case of the MLP, the most common activation function used is the *sigmoid* or the *logistic function*[4]. The logistic function is often used to model the probability that the sample $\mathbf{x}^{(i)}$ belongs to the positive class in binary classification:

$$\phi_{logistic}(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}. \tag{3.20}$$

The Softmax function [40] is a generalisation of the logistic function to be used in case of multi-class classification. The output of the Softmax function can be used to represent a categorical distribution, that is, a probability distribution over K different possible outcomes. In fact, this activation is usually combined with the *categorical cross-entropy* loss function [57, 58] for multi-class learning[5]. The formulation of the Softmax is:

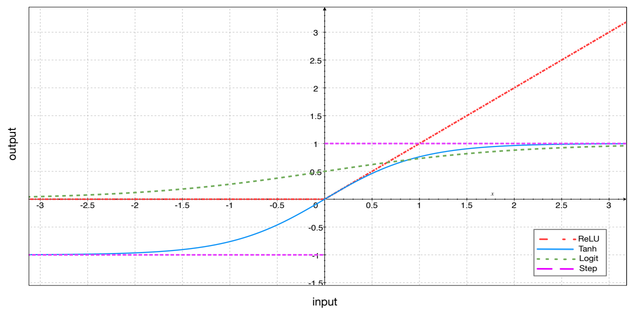
$$\phi_{Softmax}(z^{(i)})_k = \frac{e^{z_k^{(i)}}}{\sum_{j=1}^{K} e^{z_j}} \tag{3.21}$$

The categorical cross-entropy is defined as:

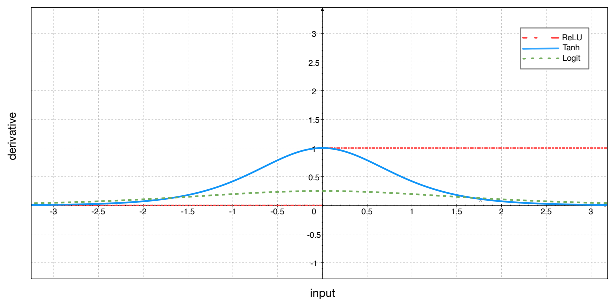
$$H(y,\hat{y}) = \sum_i y_i \log\frac{1}{\hat{y}_i} = -\sum_i y_i \log \hat{y}_i \tag{3.22}$$

[4] Although the two terms are not actually synonyms, in the literature they are used interchangeably. What is commonly referred to as the sigmoid function is in fact a special case of a sigmoid, namely the logistic function [57].

[5] The logistic activation is typically combined with the *binary cross-entropy* activation [58].



(a)



(b)

FIGURE 3.10 Activation functions (a): ReLU, tanh, Logistic, step, and their derivatives (b).

The logistic activations can be problematic in case there is a high number of negative inputs, since the output would be close to zero. This would result in a very slow training of the Neural Network, increasing the probability for the network to get stuck in a local minimum [57]. For this reason, the *hyperbolic tangent*, also known as tanh, is often preferred as an activation function for the hidden layers [55].

The tanh activation function can be interpreted as a rescaled version of the logistic function:

$$\phi_{tanh}(z^{(i)}) = 2 \times \phi_{logistic}(2 \times z^{(i)}) - 1 = \frac{e^{z^{(i)}} - e^{-z^{(i)}}}{e^{z^{(i)}} + e^{-z^{(i)}}} \tag{3.23}$$

Thus:

$$\phi_{tanh}(z^{(i)}) = \frac{1 - e^{-2z^{(i)}}}{1 + e^{-2z^{(i)}}} \tag{3.24}$$

The advantage of the hyperbolic tangent over the logistic function is that it has a broader output spectrum and ranges to the open interval $(-1, 1)$, which can improve the convergence of the back-propagation algorithm [55].

As can be seen in Figure 3.10(a) the shapes of the two sigmoidal curves look similar; however, the tanh function has a two-times larger output space than the logistic function. The same is reflected in the corresponding derivatives (Figure 3.10(b).)

Another common function used in Neural Networksis the *Rectified Linear Unit* (ReLU) activation [59]. A ReLU computes a linear function of the inputs, and outputs the result if it is positive, and 0 otherwise (see Figure 3.10):

$$\phi_{ReLU}(z^{(i)}) = \max(0, z^{(i)}) \tag{3.25}$$

The advantages of the ReLU activation are many:

- Sparse activation: in randomly initialised network, only about 50% of hidden units are activated (i.e. having a non-zero output).
- Efficient computation
- Scale-invariant: $\max(0, ax) = a \max(0, x)$.

Furthermore, this function does not suffer from the *vanishing gradient* problem [57], making the ReLU activation the *de-facto* standard for most of the popular Deep Learning models in literature [60]. The vanishing gradient problem is a difficulty affecting the training of ANN with gradient-based learning methods and back-propagation. As each of the Neural Network weights receives an update proportional to the gradient of the error with respect to the current weight, in some cases it may happen that this gradient is vanishingly small, effectively preventing the weight from changing at all. In the worst case, this may completely stop the Neural Network from further training.

3.4 Convolutional Neural Networks

This work will focus on applications of Convolutional Neural Networks (CNNs), which have been highly successful in the field of computer vision for image classification and other tasks [60, 61]. Although the usage described in this thesis (in particular in Chapter 4) will not be the analysis of images, the description of the main features of such models will refer to this most common field of application of CNNs. Their current widespread application is due to recent work, in which deep CNNs have redefined the state-of-the-art in image classification and segmentation [62, 63]. In these studies, it was found that the visual cortex contains simple cells, which are sensitive to edge-like features within small regions of the retina, and complex cells, which are receptive to collections of simple cells and are sensitive to position-independent edge-like features. These structures can be modelled by performing discrete convolutions to extract simple features across the visual field [64]. CNNs mimic this structure using a series of convolutional layers that extract a set of features from the input image, and pooling layers that perform dimensionality reduction, extract more global features and add translational invariance. The resulting output image is known as a *feature map* (Figure 3.11). At early stages, the feature maps often resemble the original image with certain elements emphasised, but they become more abstract at later stages of the network.

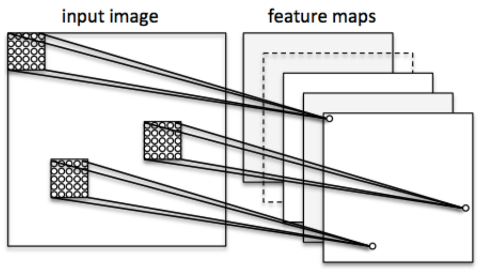


FIGURE 3.11 Feature maps representation

Since each convolutional layer generates many feature maps which have comparable dimensions to the original input image, the memory requirements and number of operations needed to evaluate the network can grow dramatically. To tackle this problem, pooling is useful since it also down-samples the size of feature maps, in a way that depends on the technique applied. e.g. max pooling and average pooling [60]. In $n \times m$ max pooling, the image is down-sampled by replacing an $n \times m$ region of the image with a single value corresponding to the maximum value in that region; in average pooling the average value is used. The pooled regions may be chosen to overlap [30, 31] to reduce information loss. Since each pixel after pooling corresponds to $n \times m$ before pooling, small translations in input features result in identical output. This decreases the network's sensitivity to the absolute location of elements in the image. Pooling can also be crucial in the extraction of macroscopic features: i.e. if a complex structure is presented to the network, like the "image" of a particle propagating in space, and only the direction of the particle is needed, pooling can help removing the background (unwanted fluctuations that could lead to misreconstruction of the direction) and obtain a cleaner image. The actual need for downsampling is related to the specific task the model is designed for. A typical example of CNN architecture is reported in Figure 3.12.

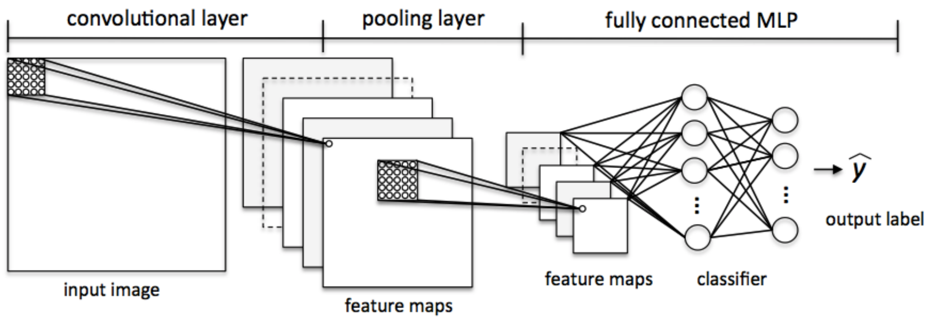


FIGURE 3.12 Example of CNN architecture

3.4.1 Convolutional arithmetics

In this section, a brief description of the convolutional arithmetics, as described in [65] will be presented. A convolutional layer's output shape is affected by the shape of its input as well as the choice of some important properties, like the *kernel* shape, the *zero padding* and the *strides* step, whose role will be explained in the following sections. This aspect is different from the case of fully-connected layers, whose output size is independent of the input size. The main concept at the basis of Convolutional Neural Networks is *affine transformations*: an input vector is multiplied with a matrix to produce an output (to which a bias vector is usually added before passing the result through a nonlinearity). This is applicable to any type of input, be it an image, a sound clip or an unordered collection of features: whatever their dimensionality, their representation can always be flattened into a vector before the transformation. All of the above mentioned input data have an intrinsic structure. In other words, they share the following important properties:

- They are stored as multi-dimensional arrays.
- They feature one or more axes for which ordering matters (e.g., width and height axes for an image, time axis for a sound clip).
- One axis, called the channel axis, is used to access different views of the data (e.g., the red, green and blue channels of a coloured image).

When an affine transformation is applied, all the axes are treated in the same way and the topological information is not taken into account. Still, taking advantage of the implicit structure of the data may prove very handy in solving some tasks, and it is convenient to preserve it. A discrete convolution is a linear transformation that preserves this notion of ordering. It is sparse (only a few input units contribute to a given output unit) and reuses parameters (the same weights are applied to multiple locations in the input).

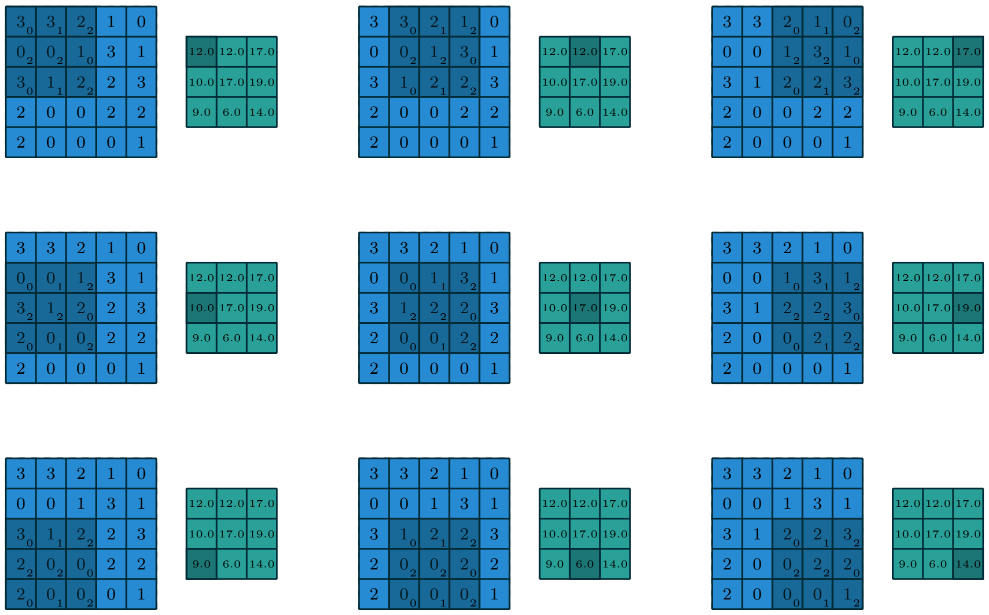


FIGURE 3.13 Computing the output values of a $2D$ discrete convolution.

Figure 3.13 provides an example of a discrete convolution: the light blue grid is called *input feature map*. For the sake of simplicity, a single input feature map is represented, but it is not uncommon to have multiple stacked feature maps. A *kernel* (shaded area) of chosen value slides across the input feature map: at each step, the product between each element of the kernel and the input element it overlaps is computed and the results are summed up to obtain the output in the current location. The procedure can be repeated using different kernels to form as many output feature maps as desired. The final outputs of this procedure are called *output feature maps*. If there are multiple input feature maps, each feature map will be convolved with a distinct kernel - and the resulting feature maps will be summed up element-wise to produce the output feature map. The convolution depicted in Figure 3.13 is an example of a 2D convolution, but it can be generalised to N-D convolutions: for instance, in a 3D convolution, the kernel would be a *cuboid* and would slide across the height, width and depth of the input feature map. The collection of kernels defining a discrete convolution has a shape corresponding to some permutation of $(n, m, k_1, \ldots, k_N)$, where n represents the number of output feature maps, m is the number of input feature maps, k_j is the kernel size along axis j.

The output size of a convolutional layer along axis j, namely o_j, is affected by the following properties:

- i_j: input size along axis j,
- k_j: kernel size along axis j,
- s_j: stride (distance between 2 consecutive positions of the kernel) along axis j,
- p_j: zero padding (number of zeros concatenated at the beginning and at the end of an axis) along axis j.

The analysis of the relationship between convolutional layer properties is made easier by the fact that they don't interact across axes, i.e., the choice of kernel size, stride and zero padding along axis j only affects the output size of axis j. For this reason, the following simplified settings will be considered in this description:

- 2-D discrete convolutions ($N = 2$),
- square inputs ($i_1 = i_2 = i$),
- square kernel size ($k_1 = k_2 = k$),
- same strides along both axes ($s_1 = s_2 = s$),
- same zero padding along both axes ($p_1 = p_2 = p$).

This facilitates the analysis and the visualisation, but the results can generalise to the N-D and non-square cases easily.

The simplest case to analyse is that in which the kernel just slides across every position of the input (i.e., $s = 1$ and $p = 0$). One way of defining the output size in this case would be by computing the number of possible placements of the kernel on the input. For instance, let us consider the width axis: the kernel starts on the leftmost part of the input feature map and slides by steps of one until it touches the right side of the input. The size of the output will be equal to the number of steps made, plus one, accounting for the initial position of the kernel. The same logic applies for the height axis. More formally: for any i and k, and for $s = 1$ and $p = 0$,

$$o = (i - k) + 1. \tag{3.26}$$

(Half) same Padding

Sometimes zero padding is applied, i.e. padding the input volume with zeros around the border to control the output shape. In general, zero padding with p zeros changes the effective input size from i to $i + 2p$. So, in the most general case, the relation between input and output shapes is: for any i, k and p, and for $s = 1$,

$$o = (i - k) + 2p + 1. \tag{3.27}$$

In some applications, having the output size equal to the input size (i.e., $o = i$) can be a desirable property. In this case, the relation between the input and output size becomes the following. For any i and for k odd ($k = 2n + 1, \quad n \in \mathbb{N}$), $s = 1$ and $p = \lfloor k/2 \rfloor = n$,

$$\begin{aligned} o &= i + 2\lfloor k/2 \rfloor - (k - 1) \\ &= i + 2n - 2n \\ &= i. \end{aligned} \tag{3.28}$$

This is sometimes referred to as *half* (or *same*) padding. Figure 3.14 provides an example for $i = 5$, $k = 3$ and (therefore) $p = 1$ [66].

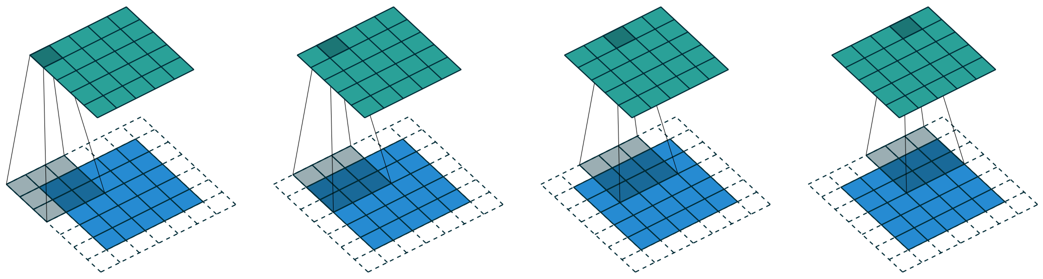


FIGURE 3.14 (Half padding, unit strides) Convolving a 3×3 kernel over a 5×5 input using half padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 1$).

Pooling arithmetic

In a neural network, pooling layers provide invariance to small translations of the input. The most common kind of pooling is *max pooling*, which consists in splitting the input in (usually non-overlapping) patches and outputting the maximum value of each patch. Other kinds of pooling exist, e.g., mean or average pooling, which all share the same idea of aggregating the input locally by applying a non-linearity to the content of some patches. Since pooling does not involve zero padding, the relationship describing the general case is as follows: For any i, k and s,

$$o = \left\lfloor \frac{i-k}{s} \right\rfloor + 1. \tag{3.29}$$

This relationship holds for any type of pooling [66]. Figure 3.15 shows as example of numerical computation of average pooling.

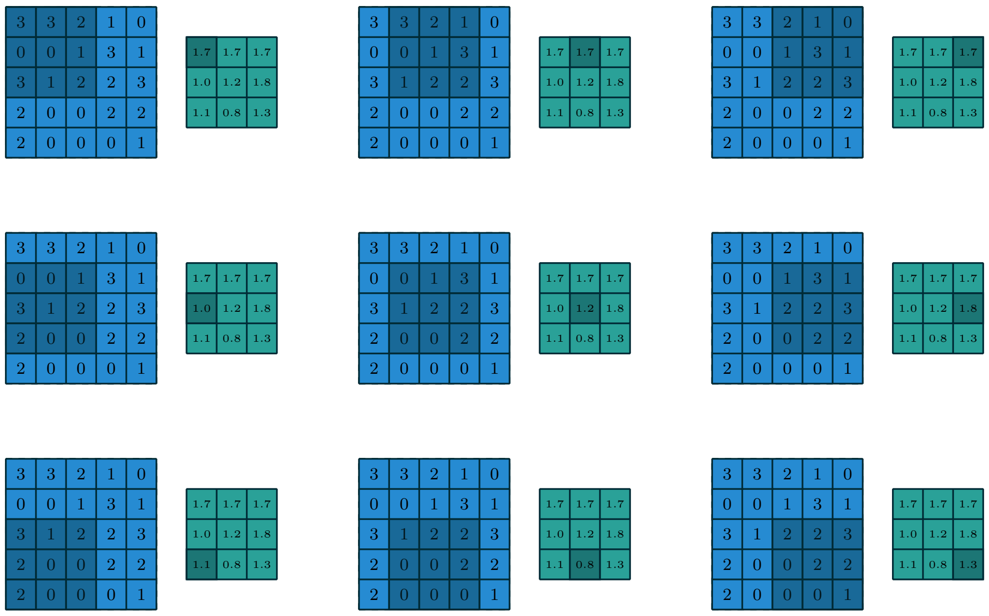


FIGURE 3.15 Computing the output values of a 3×3 average pooling operation on a 5×5 input using 1×1 strides, outputting the mean value for each patch

3.5 Evaluating Machine Learning Models

In Machine Learning, the goal is to obtain models that *generalise*, namely that perform well on new and unseen data, and *overfitting* is the central obstacle to this goal. Although it is complicated to reliably measure the generalisation power of a Machine Learning model, strategies exist for mitigating the risk of overfitting as well as maximising generalisation.

3.5.1 Training, Validation, and Test sets

Machine Learning algorithms cannot be evaluated on the same data used for the training. The reason for not doing that quickly becomes evident: the model can simply remember the whole training set, and will therefore always generate the correct prediction for any point in the training set. As a consequence, this “remembering” property does not provide any indication whether the model will be able to generalise. Consequently, model performance on unseen data is much worse compared to the one obtained on the training data. Hence, evaluating a Machine Learning model always requires the available data to be split in two disjoint sets, namely the *training* set and the *test* set. The former is used to train models, while the latter (also referred to as *hold-out* set or *external validation* set) is used exclusively to evaluate the model performances once the training process is completed.

Furthermore, Machine Learning models have several parameters and settings that can be used to control their behaviour during training. These settings, also known as *hyper-parameters*, are not automatically adapted by the algorithm during the training process, even though this could in principle be done by designing another algorithm dedicated to the parameters optimisation. Typical examples of hyper-parameters for a deep network may be the number of layers, or the size of the layers, or the different activation functions used. Hyper-parameters are usually not learnt on the training set, in order to avoid overfitting: learning the hyper-parameters on the training set might result in fitting the training set better, but at the same time not being able to correctly generalise on the test set.

To solve this problem, an additional *validation* set is usually defined. The validation set is constructed by splitting the training set in two subsets, one of which is used to learn model parameters (i.e. training), and the other one (i.e. validation) is used to estimate the generalisation error during the training process. Typically, about 80% of the total training data is used for training and a 20% for validation [57].

3.5.2 Capacity, Overfitting and Underfitting

The ability to perform well on previously unobserved inputs is called generalisation. Typically, when training a machine learning model, a training set is provided; a function to measure the error is computed and referred to as the training error, and several operations are made to minimise it. So far, it is simply an optimisation problem. What separates machine learning from optimisation is that the goal is to minimise also the *generalisation error*, also called test error, defined as the expected value of the error on a new input. Of course, when using a machine learning algorithm, the parameters (weights) are not fixed ahead of time. In fact, the training set is sampled, then used to choose the combination of parameters to reduce the training error, and then the test set is sampled. Under this process, the expected test error is greater than or equal to the expected value of training error. The factors determining how well a machine learning algorithm will perform are its ability to:

- Make the training error small.
- Make the gap between training and test error small.

These two factors correspond to the two central challenges in machine learning: underfitting and overfitting. Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set. Overfitting occurs when the gap between the training error and test error is too large, i.e. the model performs well on the training set, but fails to generalise. We can control whether a model is more likely to overfit or to underfit by altering its *capacity.* Informally, the capacity of a model is the ability to fit a wide variety of functions. Models with low capacity may struggle to fit the training set. Models with high capacity can overfit by memorising properties of the training set that do not serve them well on the test set. One way to control the capacity of a learning algorithm is by choosing its hypothesis space, the set of functions that the learning algorithm is allowed to select as the solution. For example, the linear regression algorithm has the set of all linear functions of its input as its hypothesis space. We can generalise linear regression to include polynomials, rather than just linear functions, in its hypothesis space. Doing so increases the model capacity [57].

3.5.3 Performance estimators

In order to evaluate the abilities of a machine learning algorithm, a quantitative measure of its performance must be defined. Usually the performance measurement is specific to the task being carried out by the system. For tasks such as classification, the performance is expressed in terms of the ability of the model to predict the correct label for each test sample, while for the regression problems the error in the quantity estimation is often calculated.

Accuracy

Accuracy is the fraction of examples for which the model produces the correct output [47]. If the entire set of predicted labels for a sample strictly matches with the true set of labels, then the subset accuracy is 1.0. If $\hat{y}_i$ is the predicted value of the i-th sample and y_i is the corresponding true value, then the fraction of correct predictions over N samples is defined as [67]:

$$ACC(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N-1} I(\hat{y}_i = y_i) \tag{3.30}$$

where $I(x)$ is the indicator function:

$$I(x) = \begin{cases} 1 & \text{if event x occurs} \\ 0 & \text{otherwise} \end{cases} \tag{3.31}$$

Confusion matrix

Confusion matrix is an estimator of the number of well-classified events per each class. By definition a confusion matrix C is such that $C_{i,j}$ is equal to the number of observations known to be in group i but predicted to be in group j. Thus in binary classification, the count of true negatives is $C_{0,0}$, false negatives is $C_{1,0}$, true positives is $C_{1,1}$ and false positives is $C_{0,1}$ [68]. The diagonal elements represent the number of points for which the predicted label is equal to the true label, while off-diagonal elements are those that are mislabeled by the classifier (Figure 3.16). The higher the diagonal values of the confusion matrix the better the classifier, indicating many correct predictions [47]. The confusion matrix for the classification models in this work is calculated using the function provided by `scikit-learn` metrics [67].

Receiver operating characteristic (ROC curve)

A receiver operating characteristic (ROC), or simply ROC curve, is a graphical plot which illustrates the performance of a binary classifier system as its discrimination

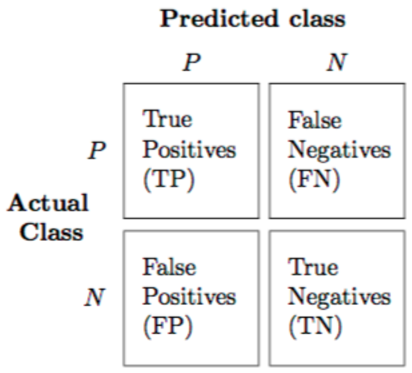


FIGURE 3.16 Confusion matrix general definition

threshold is varied. It is created by plotting the fraction of true positives out of the positives (TPR = true positive rate) against the fraction of false positives out of the negatives (FPR = false positive rate), at various threshold settings [68]. TPR is also known as sensitivity, and FPR is one minus the specificity or true negative rate (TNR), i.e. the proportion of negatives that are correctly identified as such:

$$TNR = \frac{TN}{TN + FP}, \tag{3.32}$$

where TN indicates the number of true negatives and FP represents the number of false positives. Compared to metrics such as the subset accuracy, the ROC doesn't require optimising a threshold for each label. By computing the area under the ROC curve, which is also denoted as AUC or AUROC, the curve information is summarised in one number. The diagonal of an ROC graph can be interpreted as random guessing, and classification models that fall below the diagonal are considered worse than random guessing [47]. A perfect classifier would reach the top-left corner of the graph, with a true positive rate of 1 and a false positive rate of 0. Reporting the performance of a classifier as the ROC AUC can provide further insights in a classifier's performance with respect to imbalanced samples[6].

Mean squared error (MSE)

As for regression models, one way to measure the performance of the model is to compute the mean squared error of the model on the test set. If $\hat{y}_i$ is the predicted value of the i-th sample, and y_i is the corresponding true value, then the mean squared error (MSE) estimated over N samples is defined as:

[6]However, while the accuracy score can be interpreted as a single cut-off point on a ROC curve, the ROC AUC and accuracy metrics mostly agree with each other [69]

$$MSE(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N-1} (y_i - \hat{y}_i)^2 \tag{3.33}$$

Intuitively, one can see that this error measure decreases to 0 when $\hat{y}_i = y_i$.

We can also see that $MSE(y, \hat{y}) = \frac{1}{N} \|(y_i - \hat{y}_i)\|^2$, so the error increases whenever the Euclidean distance between the predictions and the target increases [47, 68].

Coefficient of determination r^2

Sometimes it may be more useful to report the coefficient of determination r^2, which can be understood as a standardised version of the MSE. In other words, the r^2 is the fraction of response variance that is captured by the model [47]. It is defined as the ratio of the sum of the squared errors and the sum of total squares. If $\hat{y}_i$ is the predicted value of the i-th sample and y_i is the corresponding true value, then the score r^2 estimated over N samples is defined as:

$$r^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{N-1} (y_i - \hat{y}_i)^2}{\sum_{i=0}^{N-1} (y_i - \bar{y})^2} \tag{3.34}$$

where $\bar{y} = \frac{1}{N} \sum_{i=0}^{N-1} y_i$ or, in other words, it is the variance of the response. In the following, it will be rewritten as a rescaled version of the MSE:

$$r^2(y, \hat{y}) = 1 - \frac{MSE(y, \hat{y})}{Var(y)} \tag{3.35}$$

For the training dataset, r^2 is bounded between 0 and 1, with the best possible score at 1, but it can become negative for the test set (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a r^2 score of 0. If $r^2 = 1$, the model fits the data perfectly, with a corresponding MSE$=0$. In other words, r^2 provides a measure of how well future samples are likely to be predicted by the model [47, 68].

3.6 Machine Learning Applications in High Energy Physics

A key problem in experimental HEP is the correct categorisation of the particle interactions recorded by the detectors as signal and background. This characterisation is commonly performed by leveraging complex reconstruction algorithms, able to identify high-level components such as tracks, showers, bundles or clusters associated with particle interactions recorded by the detector, and summarising the energies, directions, and shapes of these objects with the extracted quantities. These quantities are then either directly selected or fed into machine learning algorithms such as Random Forest [70], K-Nearest Neighbours [71], Boosted Decision Trees [72], or Multi-Layer Perceptrons (described in Section 3.2.1) to identify particle types or separate signal from background or classify particle interactions. In the following, classical Machine Learning approaches implemented in High Energy Physics will be described.

Most High Energy Physics data analysis tasks, e.g. charged particle tracking, particle identification, signal/background discrimination, fitting and parameter estimation, normally involve several measured quantities or *feature variables*. To obtain the best possible results it is necessary to make maximal use of information in the data and employ optimal multivariate methods of analysis to perform the estimations. A classical ML pipeline includes the following steps:

- Problem definition
- Preprocessing
- Feature extraction
- Feature selection
- Learning
- Problem solution (e.g.Classification or Regression)

The key step, before the actual learning procedure, is the data preparation from preprocessing to feature selection. Whether the task is an image classification, a particle identification or the estimation of a variable, the data need to be processed in order to be fed into the Machine Learning algorithm. This process is often called *feature engineering*, and is defined as the "process of putting domain knowledge into the creation of features, to reduce the complexity of the data and make patterns more visible for learning algorithms to work" [35, 40]. This procedure is often not trivial and can be expensive in terms of time and expertise: in Machine learning, most of the applied features need to be identified by an expert and then hand-coded according to the domain and data type. For example, features can be pixel values, shapes, textures, positions and orientations - in the case of pattern recognition in images - but can also be some of the analysis parameters commonly used in fitting and reconstruction in High Energy Physics. The performances of most of the Machine Learning algorithms depend on how accurately the features are identified and extracted. Once the feature engineering is completed, several machine learning algorithms may be used to minimise the selected error function on the examples provided. In High Energy Physics experiments like KM3NeT, a common approach is to apply stacked cuts on quality parameters of reconstruction algorithms, to select the most useful pieces of information to perform the selected task, then define a set of variables (i.e. the features), which may also be a combination of multiple parameters, and leverage a Machine Learning algorithm to classify the data according to the presented examples. The variables often used for the training are reconstructed direction of the primary particle or energy flow, the distance of the event from the source position, some quality parameters of a reconstruction algorithm, such as the likelihood of the fit hypothesis, errors on the angular reconstruction, the reconstructed energy, or the number of hits related with the reconstructed track. Some applications are shown in [73] where the MLP is used to perform energy reconstruction, [74, 75], where a BDT from the tool TMVA [76] and Random Forest from `scikit-learn` [67] respectively are used to perform a signal-over-background classification and other particle identification tasks. One of the advantages of such approach is this being a rather easy way to optimise cuts and data selection; models trained are relatively simple, especially if already implemented (most packages used in HEP provide some tools to perform such kind of analyses, see for example ROOT TMVA [76]).

While these techniques have been very successful for years, they are likely to suffer some potential failures: for example, the features used to characterise the events are limited to those which have already been imagined and implemented for the experiment, and mistakes in the reconstruction of high level features from the raw data can lead to incorrect categorisation of the event. As an alternative approach, Deep Learning algorithms try to learn high-level features from raw data. This is a very distinctive aspect of Deep Learning and a major step ahead of traditional Machine Learning. Therefore, deep learning reduces the task of developing a new feature extractor for every problem [77]. Recently, computer vision has made great advances by moving away from using specifically constructed features to the extraction of features using Convolutional Neural Networks (CNNs).

Chapter 4

Deep Learning Applications for KM3NeT-ARCA

In this work, Neural Networks, and in particular Convolutional Neural Networks (simply CNN from here on), are applied to the problem of identifying and studying neutrino interactions in order to classify KM3NeT events according to their topological features in the 3D space (i.e. space-time hit distribution). In addition, these Deep Learning-based solutions have been also applied to estimate interaction parameters of neutrino events for which complex reconstruction algorithms would be needed otherwise.

Different model architectures have been designed considering the requirements of each task, by leveraging the properties of specific Neural Network layers and activation functions. The *Keras* framework, a high-level Neural Network API [58] has been used, on the *Tensorflow* [78] backend. *Keras* is one of the most popular open source framework for Deep Learning applications, capable of working on top of Theano [79], Tensorflow [78] and CNTK [80] backends, and so highly flexible to different hardware and software requirements. Within the Keras framework, common layer types are already pre-implemented and can be arranged into different architectures by specifying the desired layers and their connections and parameters. The models built, along with their applications, are described in the following sections. In more details, the DL applications presented in this Chapter are:

- up-going/down-going neutrino event classification (Section 4.2.1)
- ν_μCC/ν_eCC interaction classification (Section 4.2.2)
- neutrino energy estimation (Section 4.2.3)
- neutrino direction estimation (Section 4.2.4)

Furthermore, in Section 4.3 the estimations obtained from the defined Deep Learning models are compared to those resulting from the official reconstruction algorithm (i.e. `JGandalf`) for ν_μCC events.

4.1 Data preparation

The defined DL models take the KM3NeT triggered ν_μCC and ν_eCC events as input data. Each event is originally represented by a collection of hits, further organised as a sequence of tuples with the following structure: {`DOM-ID`, t_i}. `DOM-ID` refers to the unique identification number of the optical module that detected the signal, whilst t_i is the corresponding time instant, with ns granularity, of the i-th hit, within the 100 ms time-slice. However, this raw-level representation of the event is impractical to feed Neural Networks. In fact, input data has to be organised as a multi-dimensional array (also referred to as `ndarray` or `tensor`[1]) in which the first dimension refers to the specific event (i.e. the sample), whereas dimensions from the second to the last will stand for the considered features. In the literature, such input data tensor is usually referred to as $\mathbb{X}$, as defined in equation 3.5, where N indicates the number of samples, and M is the number of features (see Section 3.2 for more details). In the rest of this chapter, we will indicate this input tensor as $\mathbb{I}$ to avoid confusion, since X is also used to refer to the coordinate axis. Thus, considering an event as represented by a collection of hits, i.e. {`DOM-IDs`, t_i} pairs, one possible vector representation could be easily obtained by unravelling the sequence of hits of each event into a single array, then sorted by time. However, because this solution does not take into account any spatial dependency in the data, i.e. how the optical modules are distributed into the detector, a more sophisticated representation is needed.

4.1.1 Lattice definition

The so-called *detector file* defines the geometry of the detector in terms of the position and the orientation of the PMTs, the Detection Unit IDs, and the floor identification. All these positions are expressed relatively to the UTM reference point, which in the case of the ARCA detector is UTM WGS84 33N 587600 4016800 − 3450, corresponding to a latitude of 36.2922° North and a longitude of 15.975 55° East [81]. In this study, only the positions and the IDs of the DOMs hit are used. In particular, the

[1]From here on, the terms multi-dimensional array, `ndarray`, and `tensor` will be used interchangeably, as synonyms.

positions of the DOMs are determined by the centroid of the 31 PMTs contained in each optical module. In this way, each of the 2070 unique DOM IDs is then unambiguously associated to a single position in space. Therefore, considering a coordinate system whose origin is in the centre of the detector (with respect to the X and Y axes, while axis Z originates from the sea floor, pointing upwards), the positions of the DOMs are then mapped to the ranges of values reported in Table 4.1.

Axis	**min(m)**	**max(m)**
X	−451.0	494.0
Y	−470.0	474.0
Z	99.96	711.96

TABLE 4.1 X, Y, Z ranges (in metres) for the real detector.

Figure 4.1 shows a 3D view of the 115 Detection Units arranged to form one ARCA building block. Although this model is able to reproduce the spatial configuration of the real detector, it should not be used yet as it is to represent the input data. In fact, the spacing between any pair of adjacent DUs is approximately 90 m, leading to a 3D structure that is not regular. This can be better seen considering the detector footprint represented in Figure 4.2.

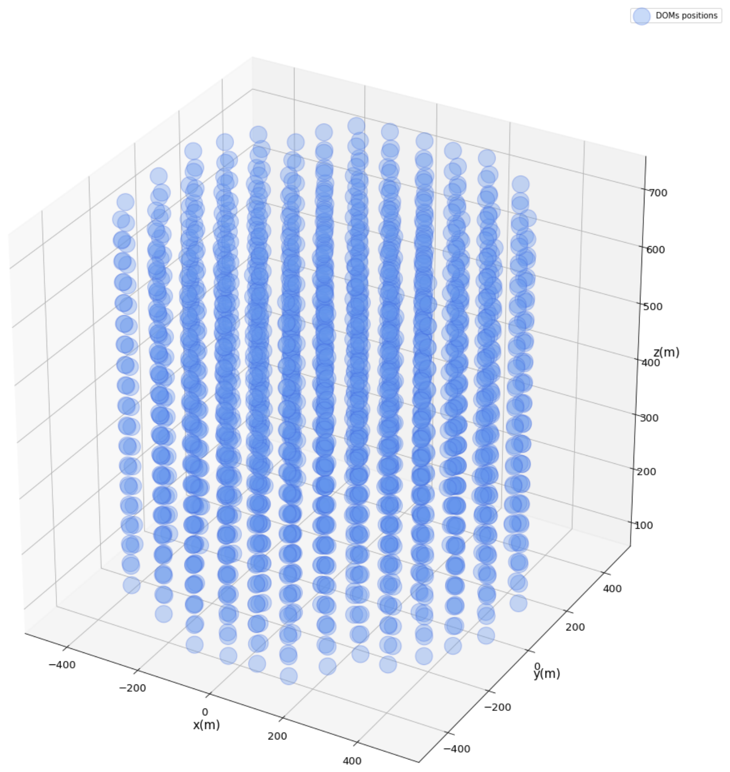


FIGURE 4.1 Detector 3D view, in which each blue circle represents the exact position of each DOM.

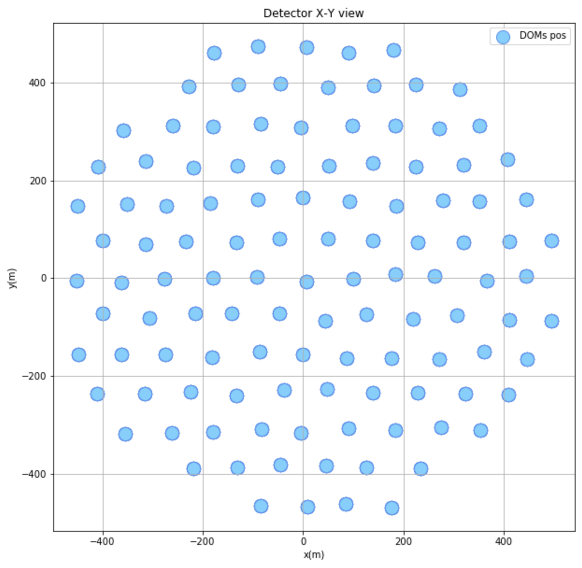


FIGURE 4.2 Detector 2D view, that is the projection on the X-Y axis of DOMs positions.

Moreover, the ordering of the strings as reported in the detector description file, namely the real order in which the DUs are being deployed at sea, is not regular as well. Figure 4.3 shows the X-Y view of the detector in which the DUs are connected following their original ordering, starting from the DU in the centre.

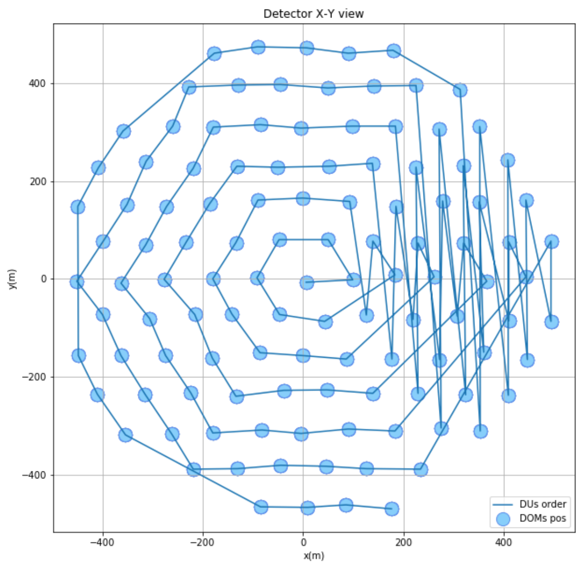


FIGURE 4.3 X-Y view of one ARCA building block with DUs connected according to their original ordering.

This aspect has to be taken into account in case data augmentation techniques (including implicit augmentation through convolution) are applied to the data to improve the effectiveness of the Machine Learning model[2]. To overcome these difficulties induced by the irregularities in the real detector geometry, a regular 3D lattice has been defined, in which the detector will then be represented.

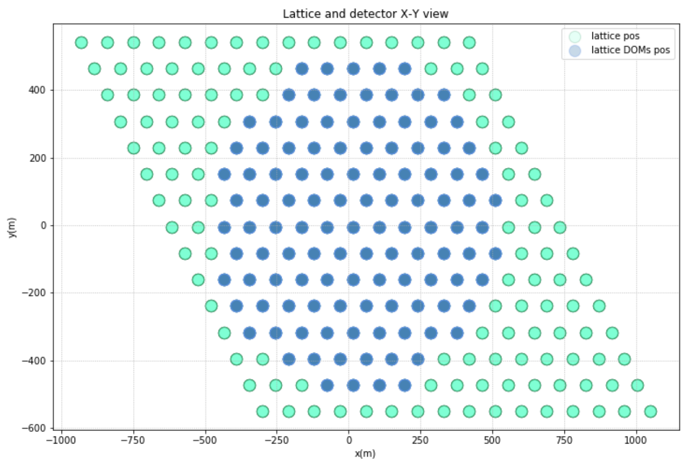


FIGURE 4.4 X-Y view of the lattice containing the "regularised" detector

As a consequence, the real positions of the DUs, and so of each DOM, have to be recalculated, so that the resulting shape of the detector would still resemble the original one. These new positions have been determined using the *k-d Tree* algorithm [82, 83], using the implementation available in the *Scipy* Python library [84]. This algorithm takes as input the positions of each DOM and the structure of the lattice, and returns the nearest neighbour for each optical module within the lattice, namely the *lattice-DOMs*. Figure 4.4 shows the footprint of the “regularised” detector as contained in the 3D regular lattice, in which the dots are exactly 90 m-spaced on the X and Y axes, and 36 m-spaced on the Z axis.

A map to connect each DOM to the corresponding *lattice DOM* is then defined, and used to collect the *lattice DOMs* hit for every event. As a matter of fact, the shifting of the coordinates in the regularised lattice structure has a relatively small impact on the DOM positions (see the distributions of the displacements between the real DOM positions and the lattice DOM positions for each coordinate in Figure 4.5, and the X-Y view of the coordinates of the real DOM and *lattice DOM* positions in Figure 4.6), but constitutes a more convenient way to represent the data as input for ML/DL learning algorithms. For example, this regular structure would allow performing convolutional

[2]For example, producing a translated version of an event by shifting the hits according to the DU number would result in events deformed after the transformation.

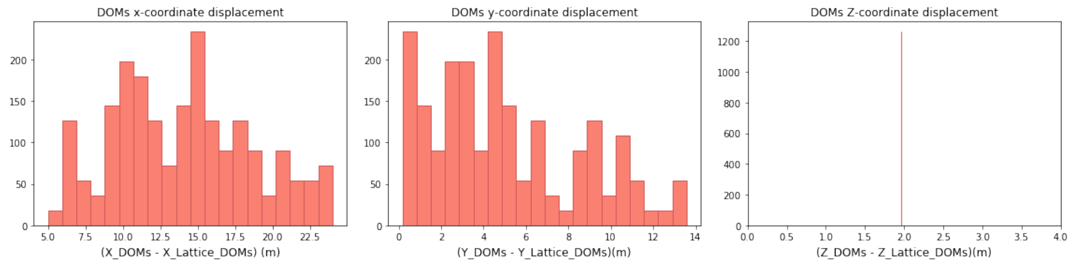


FIGURE 4.5 From left to right: distribution of the displacement between the real positions of the DOMs and the lattice position, for the x, y and z coordinates respectively.

operations by shifting the kernels, without losing information on the optical modules near the edges of the detector. Moreover, this representation also reduces the sparsity in the data, which is typically of paramount importance for ML models. Convolutions suffer from the same representation problem of data augmentation, namely the details of DOM positions are lost, which can have an impact on pointing accuracy.

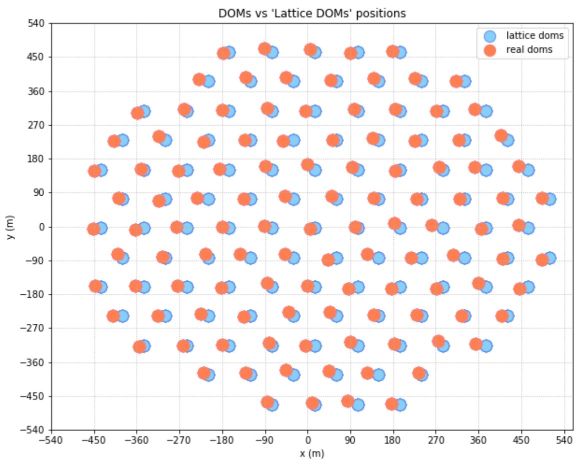


FIGURE 4.6 X-Y view of the positions of the DOMs and the lattice positions of the DOMs. The shift is relatively small, but the resulting lattice is regular.

4.1.2 Event Definition

Within the regularised detector structure, an event is represented as a 4-dimensional tensor, containing the (x', y', z') coordinates of the lattice-DOM hit, and the time t_i at which the hit occurred. Notice that the deformations on the X and Y axes are coupled, and so are the x' and y' coordinates.

In order to easily work with convolutional networks, the hit times are discretised, so that they can be easily associated to their events. In more detail, the values of the absolute time instants of the hits are binned into a chosen number of time intervals

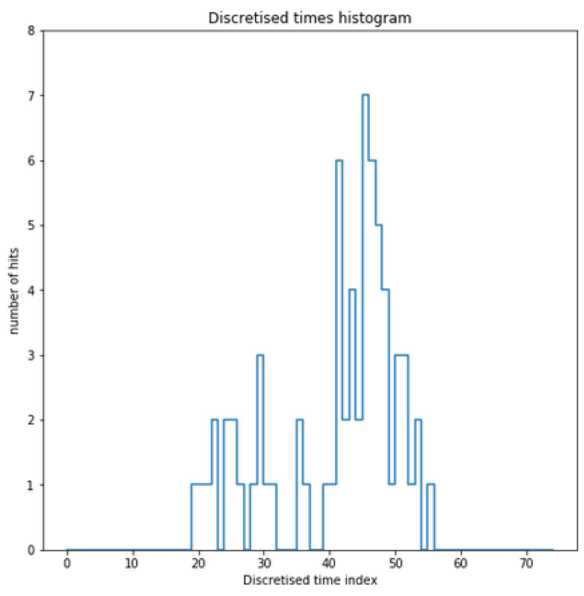


FIGURE 4.7 Histogram of the hits for a single KM3NeT event over the 75 discrete time intervals.

and then mapped to the corresponding bin indices. The discretised time array is defined starting from the minimum and maximum time instants of all events. Since most events have hit times concentrated in the centre of the timeslice, the number of bins must be chosen carefully, in order not to have a high number of hits concentrated in few (central) bins. Different configurations have been tested, and the value for the number of time bins resulting in best network performances is 75. Figure 4.7 shows the distributions of the hit times of a single event for each of the 75 bins. In this way, the collection of hits of each event is then expressed in terms of space-time integer values within a tensor that corresponds to the space-time lattice.

Hence, the main steps of the algorithm to collect and prepare event data to be passed to the ML/DL algorithms are:

1. Considering the initial set of events E, only those that have a number of DOMs hit greater than or equal to 5 are selected (more details on the rationale behind the choice of this selection threshold are reported in Section 2.2.2).

2. For each selected event, the IDs of the DOMs hit and the corresponding time instant values are collected.

3. The $\mathbb{T}$ vector of the 75 discrete time intervals is defined, considering the time instants of all the hits of the events selected at step 1.

4. The 5D tensor denoted as $\mathbb{I}$, which will finally hold the data for all the selected events, is created. The shape of $|\mathbb{I}|$ will be $[N \times 75 \times 16 \times 15 \times 18]$. N corresponds

to the total number of events; as 75 is the number of the considered time intervals (i.e. $|\mathbb{T}|$), the shape of the regular lattice structure is $16 \times 15 \times 18$.

5. For each event evt, and for each time interval $t \in \mathbb{T}$, the coordinates of the *lattice DOM* IDs hit are collected and recorded; then, if the time of the hit is contained in the current interval, the corresponding value of $\mathbb{I}$ (i.e. $\mathbb{I}[evt, t, x', y', z']$) is incremented according to the number of hits, otherwise it is set at 0.

It is worth mentioning that the construction of the $\mathbb{I}$ tensor as defined above somewhat keeps the shape of the events. Figure 4.8 compares the shape of three different ν events as represented in the real detector geometry (on the left-hand side), and within the regular lattice structure (on the right-hand side).

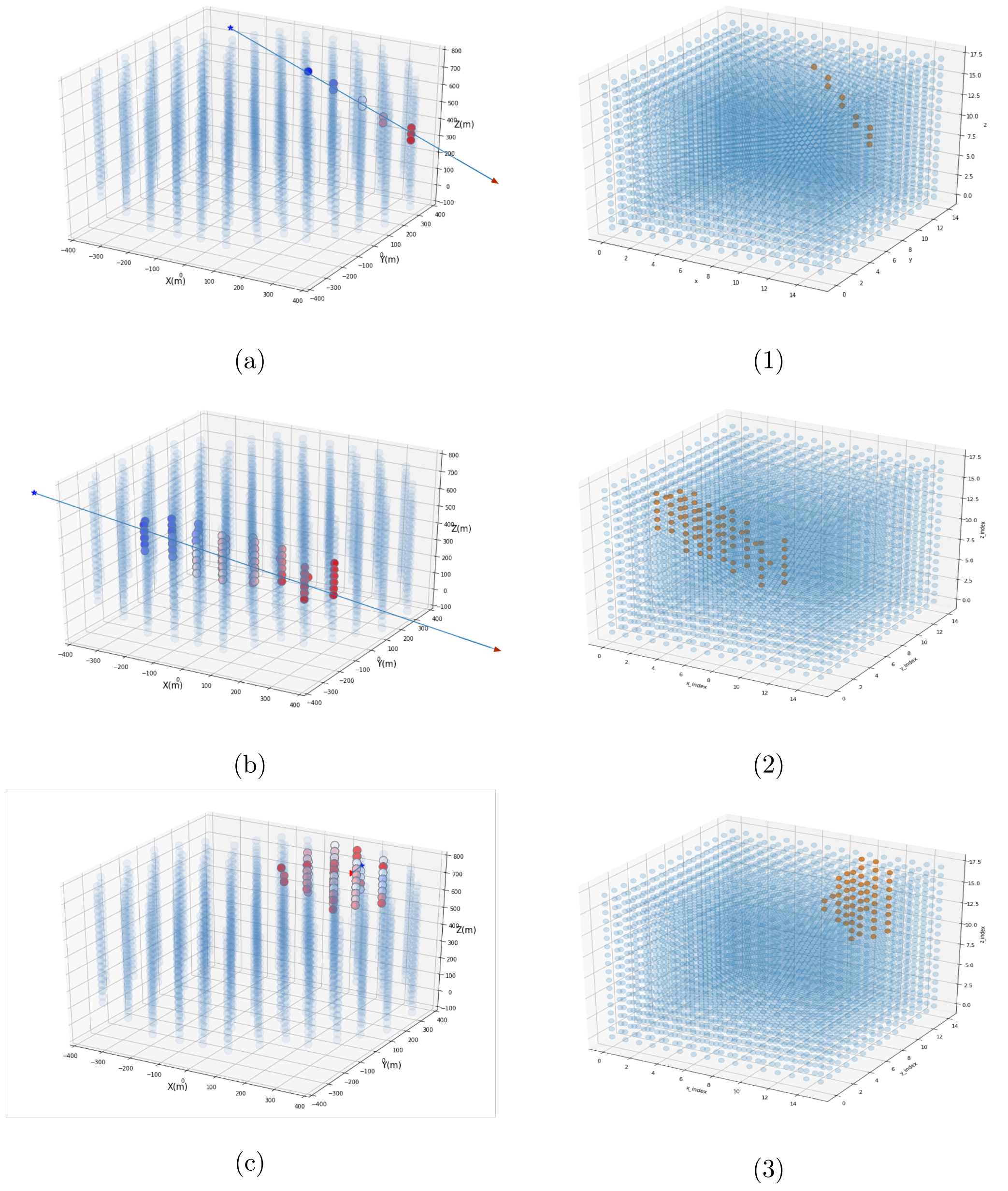


FIGURE 4.8 Event shape comparison between Detector (on the left) and Regularised Lattice (on the right), for different event topologies. Subplots (a) and (1) represent a ν_μCC track-like event; (b) and (2) a ν_μCC shower-like event; (c) and (3) an ν_eCC shower-like event.

4.1.3 Training, test and validation

A collection of $258,879$ simulated ν_μCC and ν_eCC events has been used as the reference dataset to train the DL models. This dataset (i.e. $\mathbb{I}$) is split into three subsets, namely $\mathbb{I}_{train}$, $\mathbb{I}_{val}$, and $\mathbb{I}_{test}$, so that the *training*, *validation*, and *test* sets, respectively, are generated. It is important to recall that this data splitting procedure is critical for a correct execution of the Machine Learning experimental pipeline (See Section 3.5.1 for further details).

In more details, the $\mathbb{I}$ dataset is initially split in two: $\mathbb{I}_{train}$ contains 80% of the samples, and $\mathbb{I}_{test}$ contains the remaining 20%, to be used for testing. The training set is further split in two subsets with 20% of the training samples to be associated to the $\mathbb{I}_{val}$ validation set (Figure 4.9).

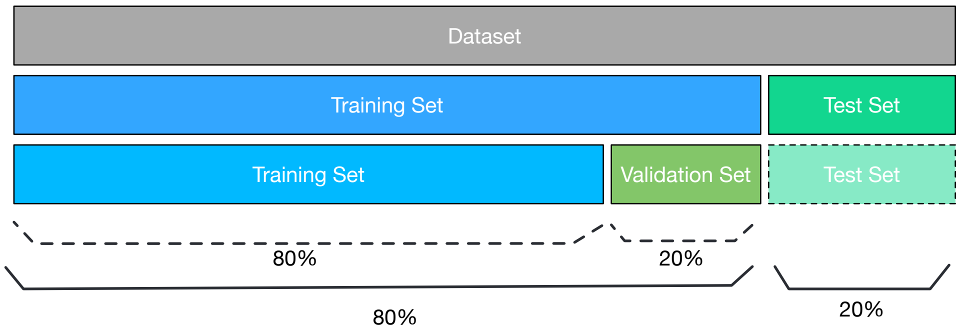


FIGURE 4.9 Sketch of how the different sets and sub-sets are defined

As a result, the total $258,879$ simulated ν_μCC and ν_eCC events are arranged into the three following datasets:

- Training and Validation set (80%): $207,061$ events, further split into:
  - $\mathbb{I}_{train}$ (80%): $165,610$ events;
  - $\mathbb{I}_{val}$ (20%): $41,451$ events;
- Test set - $\mathbb{I}_{test}$ (20%): $51,818$ events.

At each training step, the DL network evaluates its performance on the validation set, in order to monitor the evolution of the model learning during the training phase. Test data are not fed into the network until the training is completed. This common practice helps avoiding overfitting and testing the generalisation power of the network.

4.2 Learning neutrino interactions

In this Section, the defined Deep Learning solutions applied to KM3NeT neutrino interaction problems will be presented. In particular, four different tasks have been considered and analysed from the perspective of Supervised Machine Learning. The first pair of presented problems consists of two classification tasks: (1) up-going/down-going neutrino event classification (Section 4.2.1), and (2) ν_μCC/ν_eCC interaction classification (Section 4.2.2). The second two problems, on the other hand, are regression tasks: (3) neutrino energy estimation (Section 4.2.3), and (4) neutrino direction estimation (Section 4.2.4). For each of these four tasks, a Deep Convolutional Neural Networks model has been specifically designed and tested.

All the network architectures defined in this work have been originally inspired by the *VGG* model [85], which is one of the most famous CNN models excelling at the ImageNet image classification challenge [86]. The main feature of a VGG-like network is to be composed by convolutional blocks. Each block presents a couple (or more) of convolutional layers, followed by Pooling layer(s). The feature size, i.e. the first input parameter of the layer, indicating the number of convolutional filters to be learnt, is increasing from one block to the next one. This strategy to combine the multiple convolutional layers and the pooling layers, by successive increasing the number of filters in different blocks is exactly what the proposed network topologies inherits from the *VGG* model. Other hyper-parameters of the models like the size of the convolutional kernels, or the pooling strategy, and the activation functions to use, have been adapted considering the specific learning task at hand (i.e. classification or regression), as well as the size and the shape of the input features. In fact, the size of the kernels used for the applications presented in this Section is 2 to 4 times larger than those used in the original *VGG* model. Moreover, the *average pooling* strategy has replaced the *max pooling* originally used in the VGG. This is mainly because, unlike usual RGB images, the considered event snapshot "images" are sparse. Different combinations of hyper-parameters of the networks have been tested to maximise the performances for each model. The models with the best performance are shown.

In the following Sections, the defined Deep Network models will be detailed, along with the structure of the input features. Then, the obtained results will be presented and discussed. To help the understanding of the blocks composing the defined network topologies, each model scheme will follow the same conventions in representing the layers, according to the colour code shown in Figure 4.10. Further details about the specific behaviour and flavours of the single layers can be found in [58].

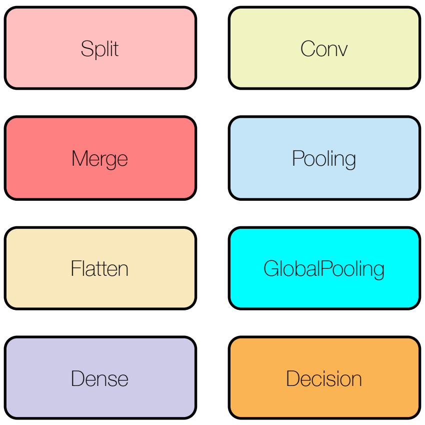


FIGURE 4.10 Colour code of the layers used as building blocks of the defined DL networks for KM3NeT. From the top left: *Split*: functional layer used to split the different views of the events; *Conv*: 2D Convolutional layer; *Merge*: functional layer used to merge the output features of multiple layers into one single layer; *Pooling*: layer to perform the Pooling operation; *Flatten*: functional layer used to reshape the (input) features of a layer - used before Dense layers; *GlobalPooling*: layer to perform the GlobalAveragePooling operation; *Dense*: Fully-Connected (a.k.a. Dense) layer; *Decision*: the very last layer of each model that performs the estimations.

4.2.1 Task 1: Up-going/Down-going neutrino event classification

The classification problem of distinguishing between up-going and down-going particles is fundamental in KM3NeT, since up-going neutrinos are the object of observation, while down-going atmospheric muons are a source of background signal. The Cherenkov light propagating in the detector volume is the main indicator of the passage of a neutrino. Even though different interactions produce events with their own signatures, the evolution along the Z coordinate axis over time indicates whether that particle is moving towards the sea surface or the sea bed. Thus, to classify events moving upwards or downwards, the evolution over time of the z-coordinate is considered.

Network architecture

Since we are interested in considering only the evolution over time along the Z axis, the input 5D tensor $\mathbb{I}$ is downscaled to form a 3D tensor. This operation is performed by summing over the X and Y axes, obtaining a 3D tensor of shape $(N, 75, 18)$[3]. An event in this representation could be visualised as a 75×18 snapshot hue image, as shown in Figures 4.11 and 4.12.

[3] N refers to the number of samples, i.e. events.

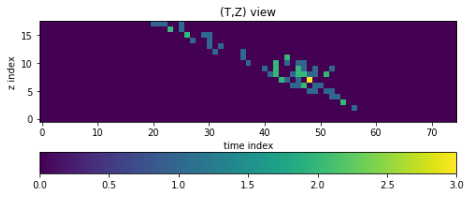


FIGURE 4.11 (T,Z) view of a down-going ν_μCC event

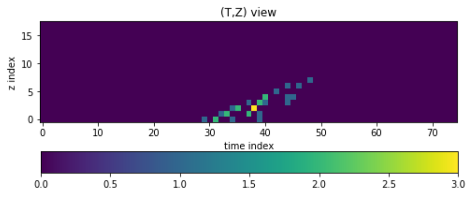


FIGURE 4.12 (T,Z) view of an up-going ν_μCC event

As for the target labels, the z component of the direction cosines, i.e. the cosine of the angle θ_z that the direction of the neutrino forms with the Z axis (also $\cos(\theta_z)$ from here on), is used to label the data. In more details, as the directions of the events are provided with respect to a coordinate system whose Z axis is pointing upwards (see Figure 4.13), the classification labels have been defined as reported in Table 4.2.

$cos(\theta_z)$	**label**
$cos(\theta_z) > 0$	$1:$ up-going
$cos(\theta_z) \leq 0$	$0:$ down-going

TABLE 4.2 Label definition for up-going/down-going classification

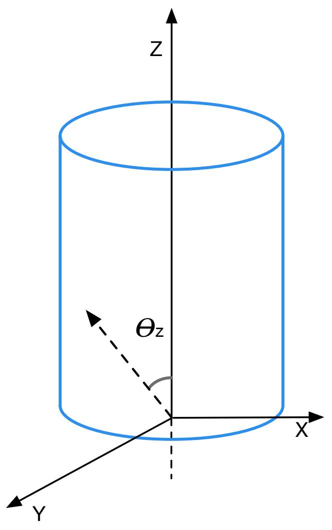


FIGURE 4.13 Coordinate system in the centre of the detector with Z axis pointing upwards. θ_z is the angle between the direction of the propagating particle and the Z axis. Therefore $cos(\theta_z) = 1$ ($\theta_z = 0°$) indicates an up-going vertical event, while $cos(\theta_z) = -1$ ($\theta_z = 180°$) indicates a down-going vertical event. The dashed arrow represents an up-going event with generic direction θ_z w.r.t. the Z axis.

The network designed for this task is composed by a stack of convolutional blocks aimed at performing local features learning (see Figure 4.14). The last convolutional module is then followed by a *Flatten* layer, i.e. an operational layer aimed at flattening the features in a way suitable for the next 2 dense layers, whose aim is to learn the global features in the data. Each convolutional module consists of 2D convolutional layers followed by an average pooling layer. The final dense layer contains a single node for each target class in the model, and a *Softmax* activation function [40] (see Section 3.3). The Softmax is applied to generate prediction values (in the $[0, 1]$ interval) for each output neuron, so that the sum of all the Softmax values will be equal to 1. The output of the Softmax function can be used to represent a categorical distribution [40]. All the defined convolutional blocks consider a kernel size of (12×12) combined with a $ReLU$ activation function [59] (also described in Section 3.3.2).

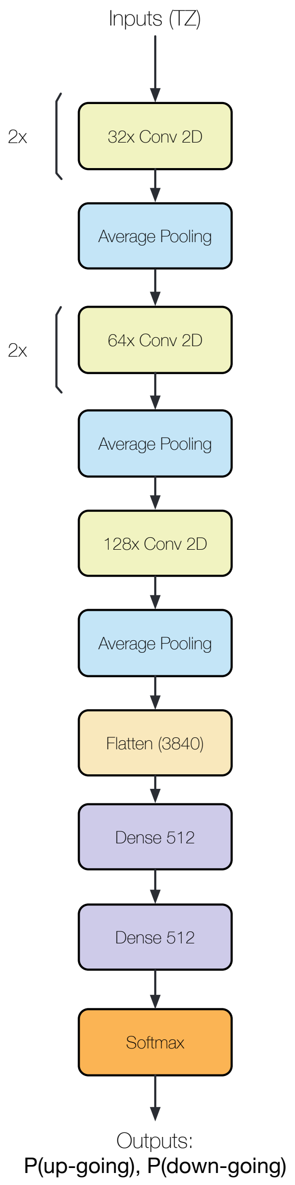


FIGURE 4.14 CNN model for upgoing/downgoing classification

Moreover, the average pooling layer applies a pooling with a size of (8×8). Padding is included in the average pooling layers to pad the input so that the output has the same shape of the original input. Average Pooling is used to reduce the computation complexity by downscaling the data and to extract low level features from neighbour neurons (see Section 3.4 for more details). For example, considering a *shower-like* event, which is widely distributed in space, its z-coordinate might globally move upwards (resp. downwards), while the corresponding collection of hits expands horizontally. Averaging the positions may help the network to extract and learn the information that the event is moving upwards or downwards, without focussing on local fluctuations (see Figure 4.15).

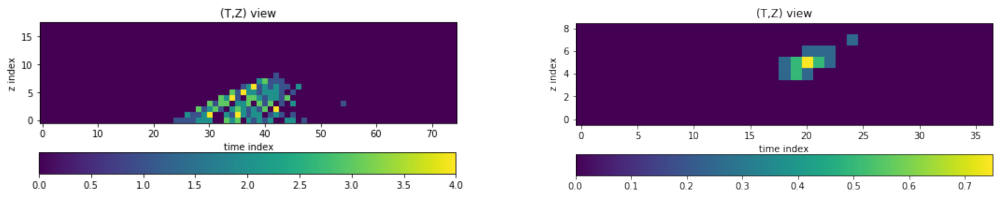


FIGURE 4.15 (T,Z) view of a ν_eCC shower event before (left) and after (right) the Average Pooling is applied.

For both training and validation, the loss function applied to measure how closely the model's predictions match the target classes, is the *categorical cross-entropy* [57] (Section 3.3).

Results and discussion

The training and validation performances of the Up-going/Down-going classification task are reported as function of the epochs (i.e. training iterations). Figures 4.16 and 4.17 show the evolution of the loss function (i.e. categorical cross-entropy) and the accuracy metric, respectively, as calculated during the training process. In particular, trends on the training and validation sets are reported.

As for the evaluation on unseen (i.e. test) data, the *accuracy* score is reported, that is the percentage ratio of correct predictions over the total number of predictions (i.e. the total number of test events):

$$ACC_{up-down} = 93.3\%$$

For a better understanding of classification performances obtained on test data, the confusion matrix is shown in Figure 4.18. It reports the number of correctly classified events, along with the misclassified ones, for each class. As shown, it follows that the percentage of misclassified events (out of the total number of test events for each

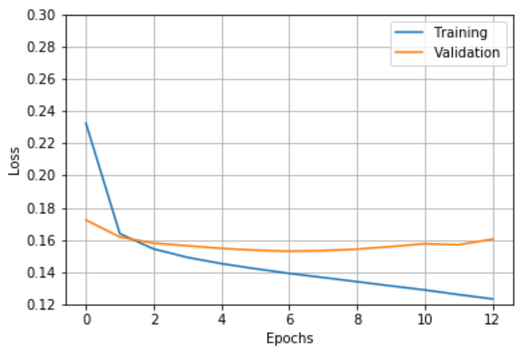


FIGURE 4.16 Loss as a function of the epochs for training and validation of Up-going/Down-going classifier

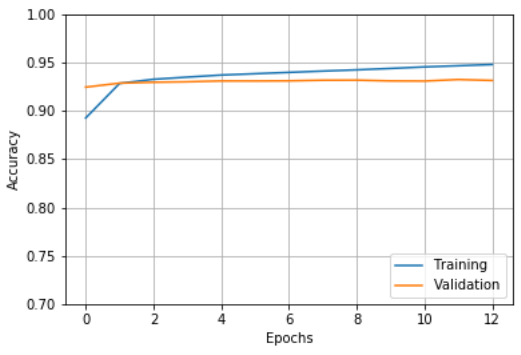


FIGURE 4.17 Accuracy as a function of the epochs for training and validation of Up-going/Down-going classifier

class) is 7.1% for down-going events, whereas for the up-going events it is 6.4%. This is compatible with the detector design, which is more efficient in the detection of up-going events.

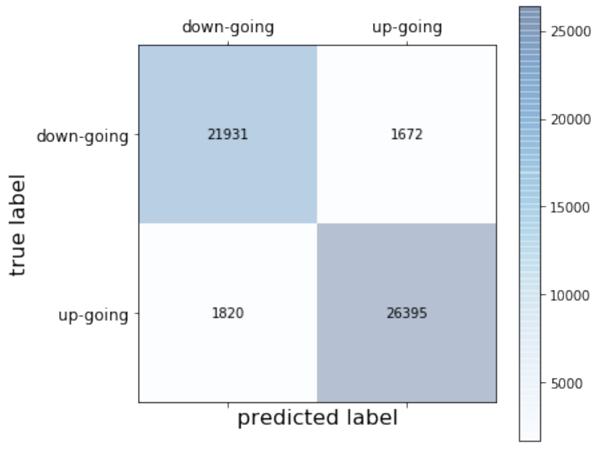


FIGURE 4.18 Confusion Matrix for Up-going/Down-going classification

The ROC curve has also been calculated as a classification performance estimator. For the whole test set, the ROC curve is shown in Figures 4.19.

Moreover, to gain even more insight on the classification performances, the ROC curve has been calculated as a function of two physical quantities of interest, namely the *neutrino energy* and the *distance of the event from the detector centre*. For each event, the distance from the detector centre is evaluated as the distance of the neutrino straight line (identified by the position and direction cosines of the propagating particle at can level) from the centre point of the detector, as illustrated in Figure 4.20.

Figure 4.21 shows the ROC curve as a function of the incoming neutrino energy: each curve shows the classification performances for events falling in a specific energy

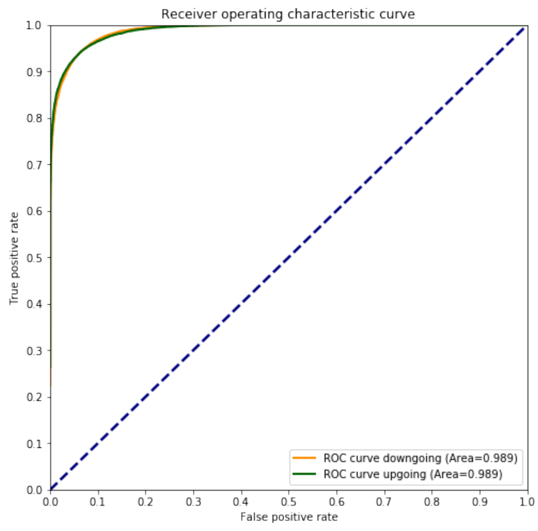


FIGURE 4.19 ROC curve for upgoing-downgoing classification

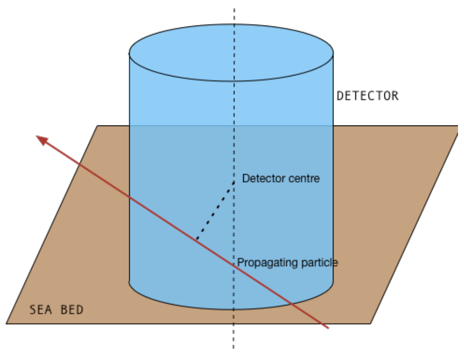


FIGURE 4.20 Definition of distance from the detector centre

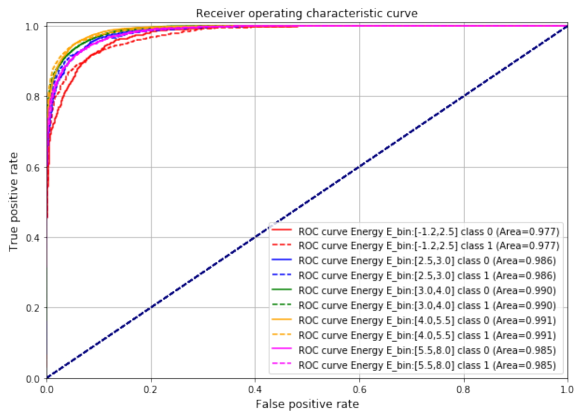


FIGURE 4.21 ROC curve for upgoing-downgoing classification as a function of the neutrino energy

range. The energy bins considered in the calculation are: $[-1.2, 2.5, 3.0, 4.0, 5.5, 7.9]$ (in terms of $log_{10}(E)[GeV]$). Similarly, 5 distance bins have been defined, namely $[0.5, 170, 320, 450, 500, 810]$, expressed in metres. The corresponding ROC curve, for each distance bin, is shown in Figure 4.22.

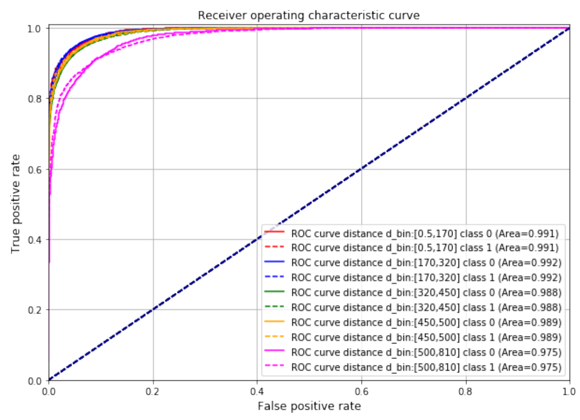


FIGURE 4.22 ROC curve for upgoing-downgoing classification as a function of the distance of the neutrino from the detector centre

As expected, the classification performances get worse for lower energies and high distances from the centre.

A possible explanation for the energy dependency is that with a small number of hits fluctuations affect the determination of the direction for particles approaching the detector nearly horizontally. At high energy the available training sample is sparse, so the network training is not optimal.

The distance dependency, on the other hand, is related to the finite volume of the detector. Events occurring on the edge (or close to it), might not be completely included in the instrumented region, and so they are more difficult to classify.

The classification efficiency is also defined as the accuracy per energy (resp. distance) bin. Figures 4.23 and Figure 4.24 show the efficiency as a function of the neutrino energy in the range 10^2-10^8 GeV and as a function of the distance of the event from the detector centre up to approximately 800 m.

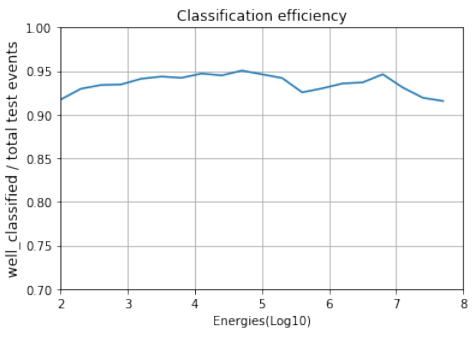


FIGURE 4.23 Classification efficiency for upgoing/downgoing events as a function of the neutrino energy

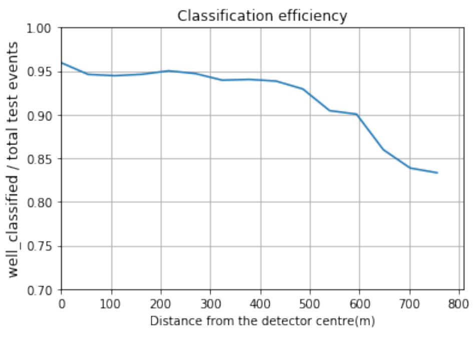


FIGURE 4.24 Classification efficiency for upgoing/downgoing events as a function of the distance of the neutrino from the detector centre

4.2.2 Task 2: ν_μCC/ν_eCC interaction classification

Events produced by muon and electron neutrino interactions can show different shapes, according to the neutrino energy and the particles produced in the interaction with the sea water. In particular, high energy muon neutrinos produce muons which result in track-like events, or showers, while charged current electron neutrino interactions result in shower-like events (See Section 1.1.2). Furthermore, the energy of the events often influences the detection, as events with few hits collected may be difficult to classify. This variability makes this kind of task suitable for Neural Network application, because the technique can be used to automatically extract features to identify the particle type.

Network architecture

Since the ν_μCC and ν_eCC interactions classification depends on the shape of the events, which is generally different for muon and electron neutrino events, for this task two views of the events are considered. In fact, unlike the up-going/down-going classification problem, in which only the evolution of the z-coordinate over time has been considered, in this task we also introduce the x/y-coordinate views as input features. Thus, the input data tensor is constructed starting from the 5D input tensor $\mathbb{I}$ containing the evolution of the 3D coordinates of the particle position over time. For the specific task, two 3-dimensional tensors are defined:

- $\mathbb{TZ}$, a tensor of shape $(N, 75, 18)$[4], obtained by summing over the X and Y axes;
- $\mathbb{XY}$, a tensor of shape $(N, 16, 15)$, obtained by summing over the T and Z axes.

As a consequence, an event in this representation in seen as two snapshot images of the corresponding shapes (see Figures 4.25-4.28 for examples).

In this task, the target labels indicate whether an event has been produced by a muon neutrino or an electron neutrino CC interaction. Therefore, from a supervised learning perspective, this is a binary classification task (see Table 4.3):

type	**label**
ν_μCC	1
ν_eCC	0

TABLE 4.3 Labels definition for ν_μCC/ν_eCC classification

[4]As above, N indicates the sample size, i.e. the number of events

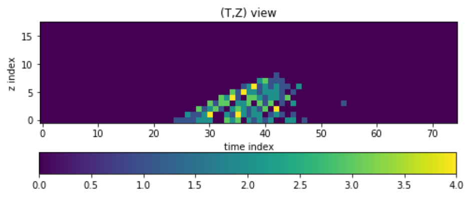


Figure 4.25 (T,Z) view of a 2.5 TeV ν_eCC event

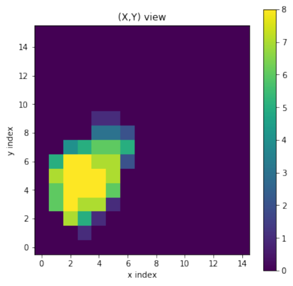


Figure 4.26 (X,Y) view of a 2.5 TeV ν_eCC event

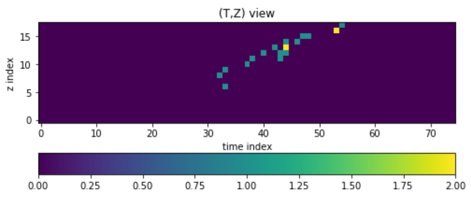


Figure 4.27 (T,Z) view of a 600 GeV ν_μCC event

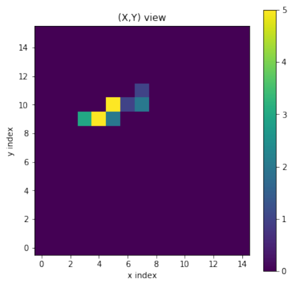


Figure 4.28 (X,Y) view of a 600 GeV ν_μCC event

The network architecture designed to tackle this classification task consists of two separated branches, corresponding to the two different views of the same event provided as input features (see Figure 4.29). These two views represent two separate perspectives to learn from for the convolutional modules. The two separate branches are further concatenated just before the last dense layers, by operational layers that in order flatten and merge the outputs of the two sub-networks.

Of these two branches, one is dedicated to learn the features from the time evolution of the z coordinate, similarly to what has been used for up-going/down-going classification. On the other hand, the second sub-network learns from the projection of the event on the (X, Y) plane. These two parallel branches have identical configurations: 3 convolutional blocks, composed by 2D convolutional layers plus *ReLU*, combined with an average pooling layer. The sizes of convolutional and pooling kernels are (12×12) and (8×8), respectively. The 2 Dense layers in the network end apply the *ReLU* activation function as well, along with the Softmax activation function to calculate the output predictions. The loss function to be optimised is the *categorical cross-entropy*.

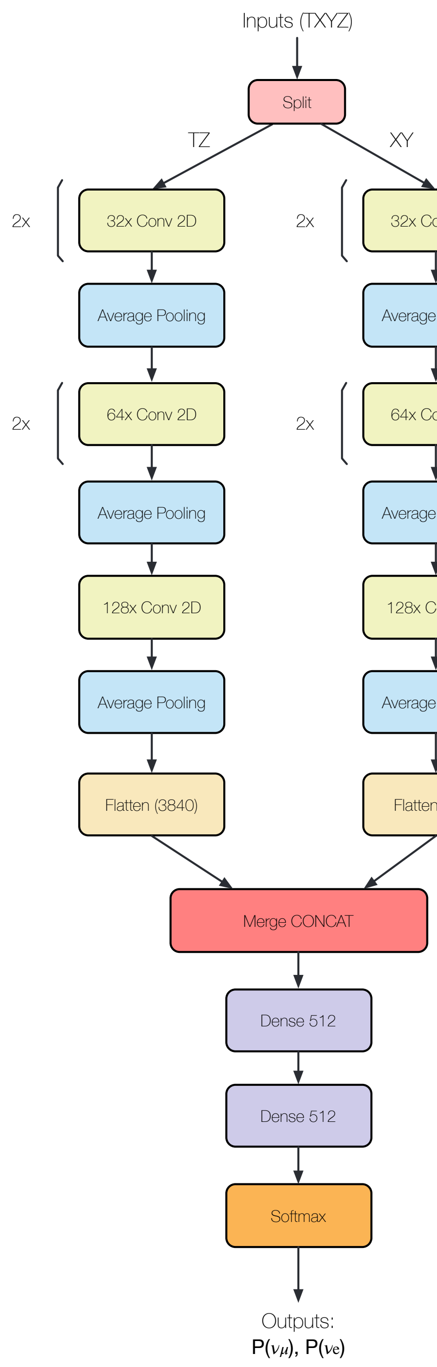


FIGURE 4.29 CNN model for ν_μCC/ν_eCC classification

Results and discussion

The training and validation performances for the ν_μCC/ν_eCC classification are shown as a function of the epochs in Figures 4.30 and 4.31.

As for the performance estimation on the test set, the accuracy score is calculated:

$$ACC_{\nu_\mu CC/\nu_e CC} = 92.8\%$$

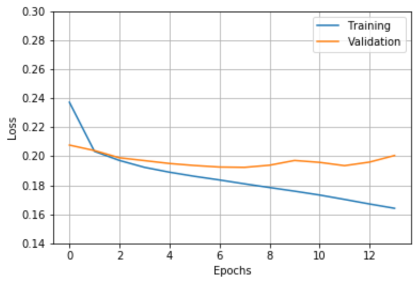


FIGURE 4.30 Training loss evolution for ν_μCC/ν_eCC classification.

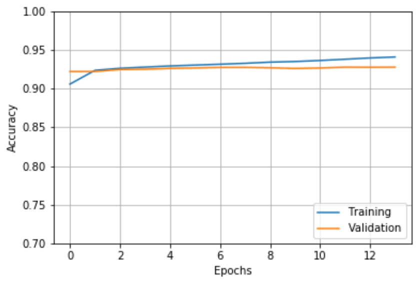


FIGURE 4.31 Training accuracy evolution for ν_μCC/ν_eCC classification.

as well as the confusion matrix (see Figure 4.32).

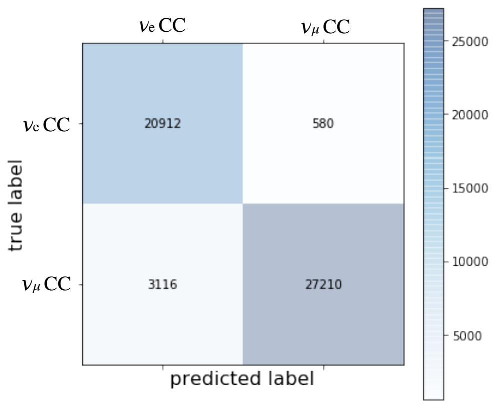


FIGURE 4.32 Confusion Matrix for ν_μCC/ν_eCC classification.

The confusion matrix reports that the percentage of misclassified events is higher in the case of ν_μCC events. In fact, 2.7% of the ν_eCC events are misclassified, whereas the same quantity is 10.3% for ν_μCC events.

For better insight on the phenomenon, the ROC curves for the whole test dataset has been computed, as well as the ROC curves for event classes split according to their neutrino energy and their distance from the detector centre. As reported in Figures 4.33, 4.34, and 4.35, the dependency on the energy and the distance is more evident for ν_μCC/ν_eCC classification than the up-going/down-going case. The reasons behind this may be due to the difficulty of discriminating ν_μCC/ν_eCC interactions at lower energies and when events occur near the detector edges.

Again, similarly to the previous classification task, the efficiency for the ν_μCC/ν_eCC interaction classification is shown in Figures 4.36, and 4.37 as a function of the neutrino energy and of the distance from the detector centre, respectively.

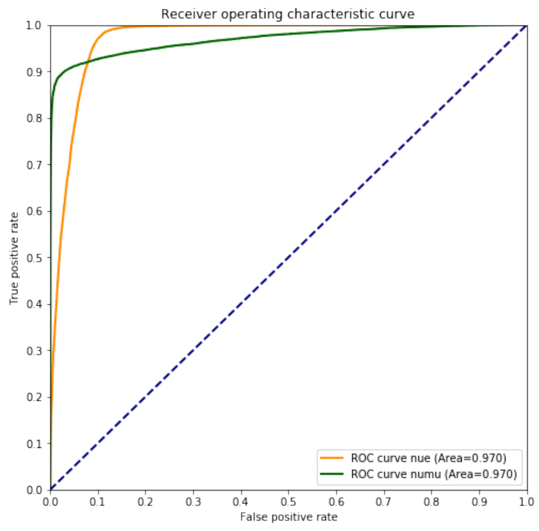


FIGURE 4.33 ROC curve for ν_μCC/ν_eCC classification.

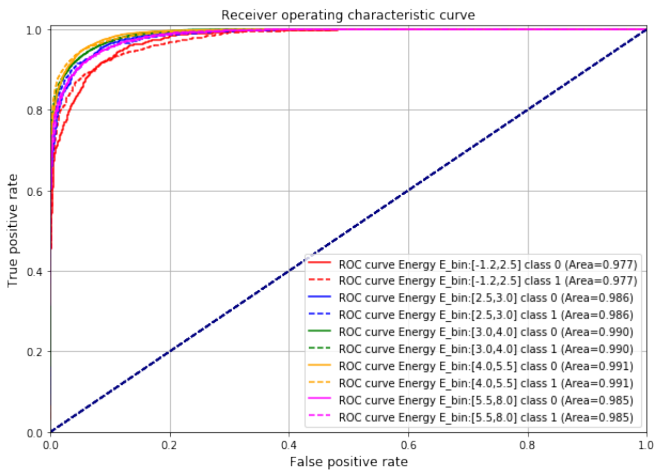


FIGURE 4.34 ROC curve for ν_μCC/ν_eCC classification as a function of the neutrino energy.

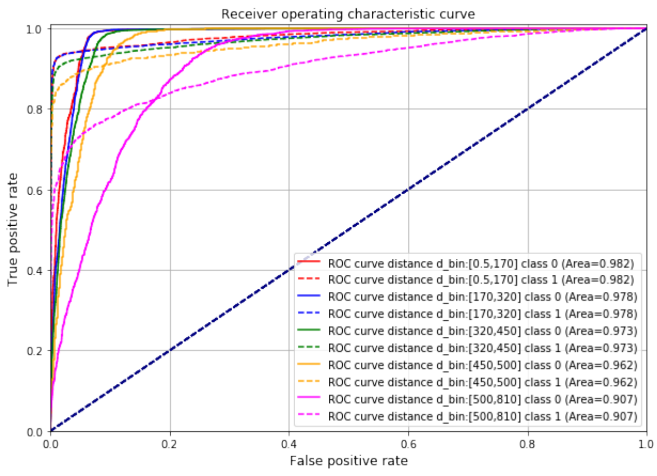


FIGURE 4.35 ROC curve for ν_μCC/ν_eCC classification as a function of the distance of the neutrino from the detector centre.

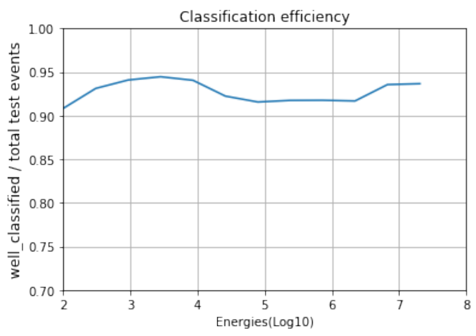


FIGURE 4.36 Classification efficiency for ν_μCC/ν_eCC events as a function of the neutrino energy

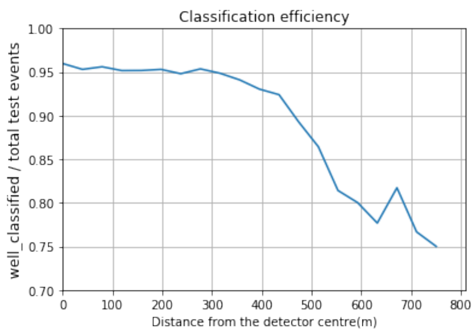


FIGURE 4.37 Classification efficiency for ν_μCC/ν_eCC events as a function of the distance of the neutrino from the detector centre

4.2.3 Task 3: Neutrino energy estimation

The estimation of the particle energy represents one of the main goals of high energy neutrino detection, as it can provide information on the spectrum and origin of the incoming neutrinos. From a simulation point of view, the energies that are recorded are those of the tracks of the simulated particles as they reach the can. Therefore, for the ν_μCC and ν_eCC events, the energies considered are those of the muon and the electron produced in the neutrino interaction, respectively. From the perspective of the learning algorithm, the estimation of the energy may be easily defined as a regression problem. As in the previous analysis, since the shapes of the events are related to their energy, the estimation cannot be performed without taking into account the event topology from multiple projections (i.e. views).

Network architecture

The input data format for this task is the same used for the classification of ν_μCC/ν_eCC interaction classification, namely two views of the same event of shapes 75×18 and 16×15, respectively. For the sake of completeness, the structure of the input features is the following:

- $\mathbb{TZ}$, a tensor of shape $(N, 75, 18)$, obtained by summing over the X and Y axes;
- $\mathbb{XY}$, a tensor of shape $(N, 16, 15)$, obtained by summing over the T and Z axes.

Similarly to other tasks, both ν_μCC and ν_eCC events have been used to train the deep neural network model. Despite the input features being the same of the ν_μCC/ν_eCC classification task, the learning targets are different, namely the values of the energy for each event. However, to aid the optimisation of the gradients, and for better numerical approximations, $\log_{10} E$ has been estimated by the neural network, rather than the values of E. To simplify the notation, in the following E and $\hat{E}$ will still be used to refer to the expected and the estimated values of energy, respectively.

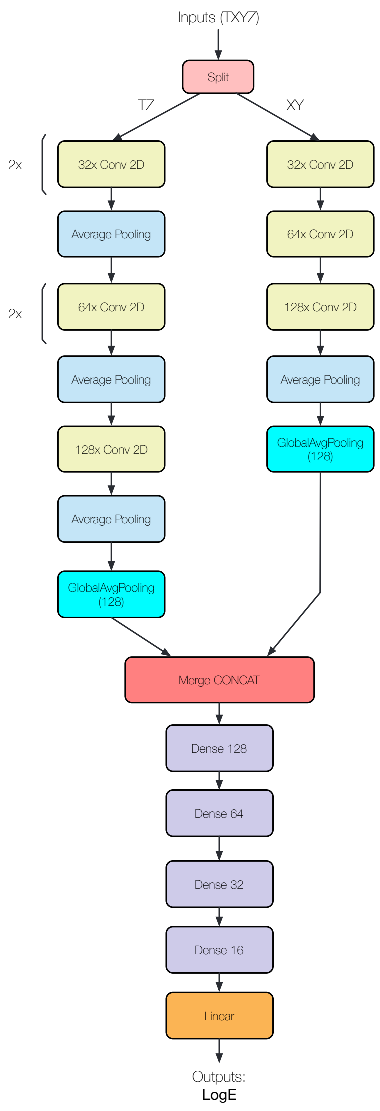


FIGURE 4.38 CNN model for neutrino energy estimation

The model architecture, shown in Figure 4.38, consists of 2 parallel sub-networks, which analyse separately the $\mathbb{TZ}$ and the $\mathbb{XY}$ tensors, merged and fed into multiple fully-connected layers to combine and extract features learnt from the multiple views of the events. In particular, the branch on the left-hand side (i.e. TZ) consists of 3 convolutional modules, where two of them include a pair of 2D convolutional layers with the same number of filters.

All these layers apply a 12×12 kernel to the data, and *ReLU* activation function. Average pooling strategy is applied after each convolutional module, using a (6×6)

pool, and stride steps equal to 2, i.e. the filter convolves around the input volume by shifting two units at a time. The sub-network ends with a *Global Average Pooling* (GAP) layer [87]. The GAP layer reduces each feature map to a single number by simply taking the average of all values, event by event. This layer learns to perform a dimensionality reduction on the data, by maximising the contributions of the most relevant features.

As for the right-hand side network branch (i.e. XY), the defined topology is smaller, as the shape of the input features, i.e. (16×15). A single convolutional block equipped with three 2D convolutional layers of increasing number of filters are defined. These layers apply kernels of size (12×12), with average pooling (with 4×4 pools), and a GAP layer in the very end. The two branches are then concatenated and their output is redirected to a module consisting of 4 fully-connected (i.e. dense) layers with hyperbolic tangent (*tanh*) as the activation function. The last prediction layer is again a dense layer with no activation function, as a single number has to be estimated. The loss function is the *mean squared error* (MSE), used with the *Adadelta* [88] optimiser. This optimiser dynamically adapts over time, using only first order information and has minimal computational overhead beyond stochastic gradient descent. This tool has been empirically proved better in the case of sparse input data, which is the case for the data at hand.

Results and discussion

For this regression task, performance are evaluated in terms of the estimation errors, i.e. how much the estimated values $\hat{E}$ differ from the expected ones, denoted as E. The evolution of the loss function during the training phase is displayed in Figure 4.39, for both the training and the validation phase.

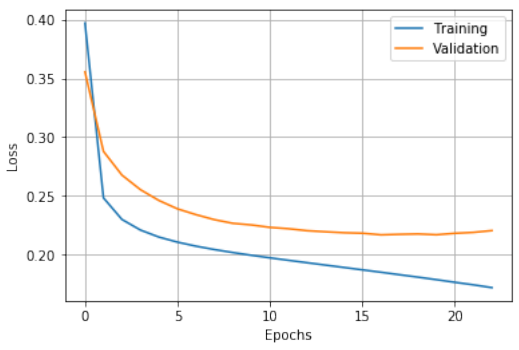


FIGURE 4.39 Energy estimation model history: Loss function evolution during training and validation

The mean squared error (MSE) for the energy estimation is:

$$MSE(E, \hat{E}) = 0.22,$$

with a r^2 score (coefficient of determination) of:

$$r^2(E, \hat{E}) = 0.84.$$

Regression performances can also be analysed by plotting estimated (log) energy values as a function of the *true* values.

The resulting scatter plot (Figure 4.40), shows that energy values accumulate along the bisection line, i.e. the cases in which estimations and expectations are identical.

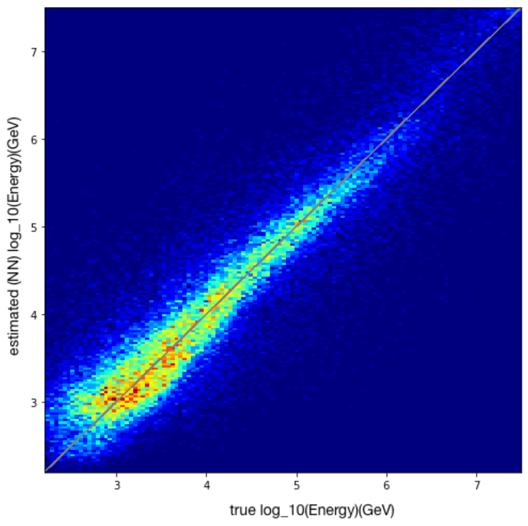


FIGURE 4.40 Regression performance on the estimation of $log_{10}(E)[GeV]$ of the leptons for ν_μCC $\cup$ ν_eCC events.

Since most of the events are concentrated in the energy region between 1 and 10 TeV, it is convenient to show the plot in a different shape, highlighting the regions corresponding to 68% and 90% confidence intervals. Figure 4.41 shows the plot in which the estimated values are reported in the X axis (whereas the expected values are in the Y axis), along with the energy resolution plot.

For a deeper understanding of the performances of the estimation, the same results can be analysed for ν_μCC and ν_eCC test events separately. The scatter plots of the regression performances and the energy resolution for both event classes are reported in Figures 4.42 and 4.43.

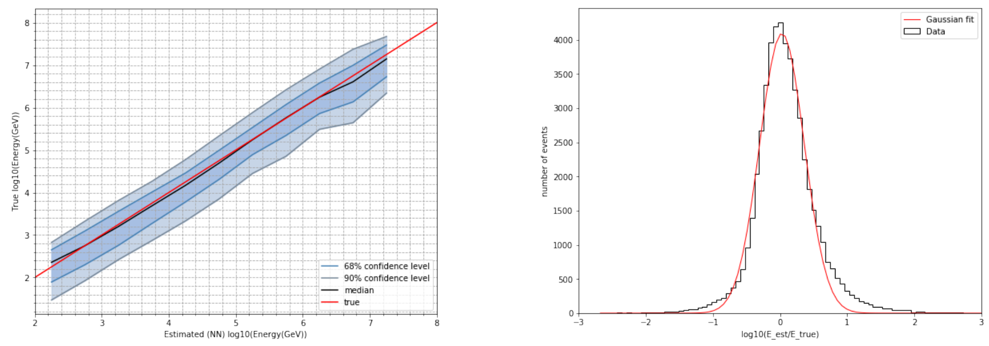


FIGURE 4.41 Left: Regression performance of energy estimation with 68% and 90% intervals on the whole (ν_μCC $\cup$ ν_eCC) test dataset. True energies are displayed as a function of the estimated values. The results shown are affected by low statistics at high energies. Right: Distribution of $\log_{10}(E_{est}/E_{true})$. The red line represents the Gaussian fit with $\mu = 0.03$ and $\sigma = 0.33$.

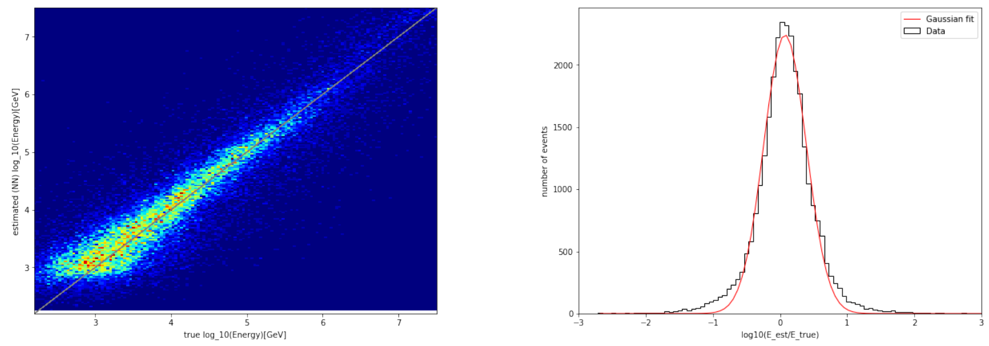


FIGURE 4.42 Left: Regression performance of energy estimation for ν_μCC event test dataset. Right: Distribution of $\log_{10}(E_{est}/E_{true})$. The red line represents the Gaussian fit with $\mu = 0.07$ and $\sigma = 0.32$.

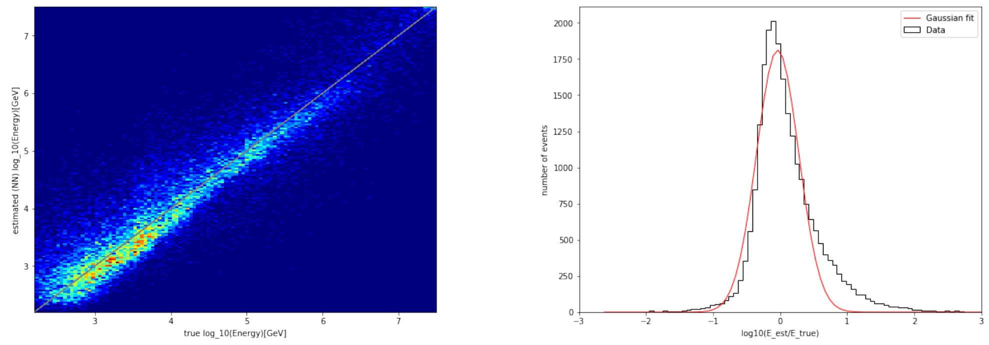


FIGURE 4.43 Left: Regression performance of energy estimation for ν_eCC event test dataset. Right: Distribution of $\log_{10}(E_{est}/E_{true})$. The red line represents the Gaussian fit with $\mu = -0.04$ and $\sigma = 0.32$.

4.2.4 Task 4: Neutrino direction estimation

Neutrino direction estimation with Neural Networks can be formalised again as a regression problem: the neural network model learns how to estimate the value of a specific variable. In this Section, the estimation of the z component of the neutrino direction cosines will be presented. Such results constitute the starting building block upon which a full direction estimation can be performed by Deep Neural Networks.

Network architecture

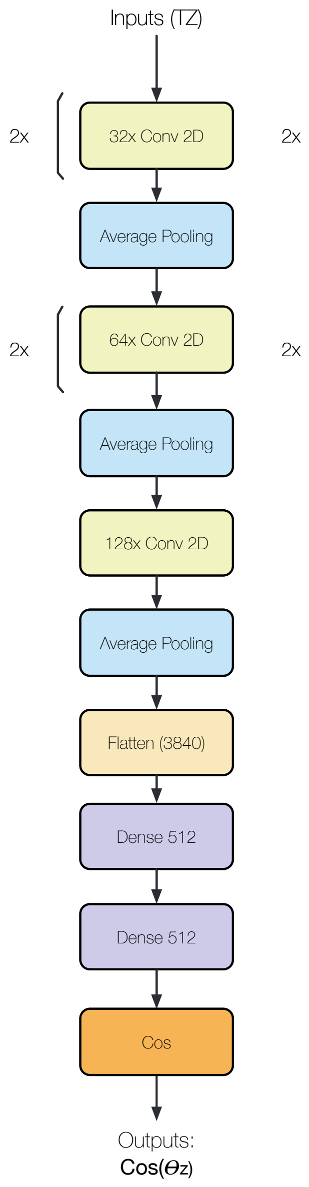


FIGURE 4.44 CNN model for $cos(\theta_z)$ estimation

The estimation of the z component of the direction cosine of events will leverage the analysis of the evolution of the z-coordinates over time as input features. Thus, the 5D $\mathbb{I}$ tensor is transformed in the 3D array of shape $= (N, 75, 18)$, by summing over the X and Y axes. As shown in Figure 4.44, the network topology is quite similar to the one adopted for up-going/down-going event classification (see Section 4.2.1). The real difference between the two models is in the activation functions used in all the layers, namely the hyperbolic tangent, *tanh*, rather than the classical $ReLU$. Moreover, the activation function in the prediction layer is the *cosine function*, since cosine values must be estimated. The loss function to minimise in this task is the mean squared error (MSE) using the Adadelta optimiser [88].

Results and discussion

As for the regression task to estimate the z-component of the direction cosine, i.e. $cos(\hat{\theta_z})$, performances are evaluated in terms of the prediction error. The MSE loss function is evaluated at each iteration during the training phase. The trend is shown in Figure 4.45 for the training and validation phases. Note that the *bump* around epoch 15 is likely caused by the optimiser used in the training phase of the model, which uses an adaptive technique to adjust the learning rate with momentum.

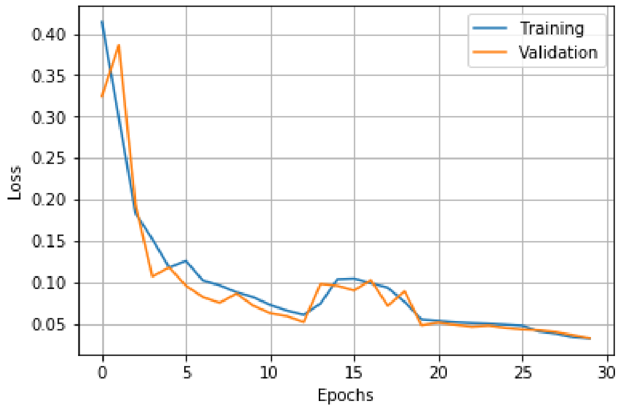


FIGURE 4.45 $cos(\theta_z)$ estimation model history: Loss function evolution during training and validation

The mean squared error (MSE) for the $cos(\theta_z)$ estimation is:

$$MSE(cos(\theta_z), cos(\hat{\theta_z})) = 0.03,$$

with an r^2 score (coefficient of determination) of:

$$r^2(cos(\theta_z), cos(\hat{\theta_z})) = 0.89.$$

The regression performances can be visualised also plotting the estimated $cos(\hat{\theta_z})$ values as a function of the *true* values. The resulting scatter plot (Figure 4.46-left), shows that estimated values gather along the diagonal curve, where $cos(\hat{\theta_z}) = cos(\theta_z)$. For better insights on the reliability of the estimations, the relationship between the true and estimated values within the regions of the 68% and 90% confidence intervals is also reported in Figure 4.46-right.

The origin of the horizontal stripes in the in the density plots (Figure 4.46) is currently not thoroughly understood, but there are hints they may be related either to the finite size of kernels and/or to the DOM top-down asymmetry (fewer PMTs looking upwards). Moreover, this effect indicates that the current choice of the kernel size is probably not suitable to analyse both ν_μCC and ν_eCC events with the same model hyperparameters. This hypothesis may be further confirmed by analysing the performance for ν_μCC and ν_eCC events separately. As reported in Figure 4.47, $cos(\theta_z)$ is better estimated for ν_μCC events, mostly track-like events, for which the direction is more clearly defined with respect to the shower case.

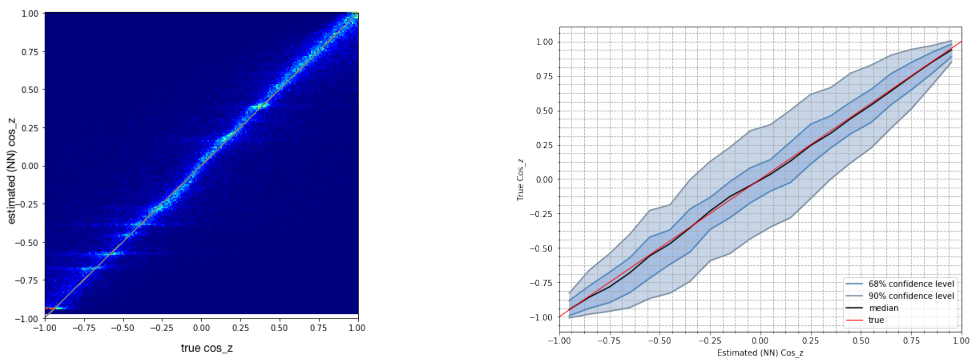


FIGURE 4.46 Left: Regression performance of $cos(\theta_z)$ estimation for ν_μCC $\cup$ ν_eCC events. Right: Regression performance of $cos(\theta_z)$ estimation with 68% and 90% intervals on the whole ν_μCC $\cup$ ν_eCC test dataset.

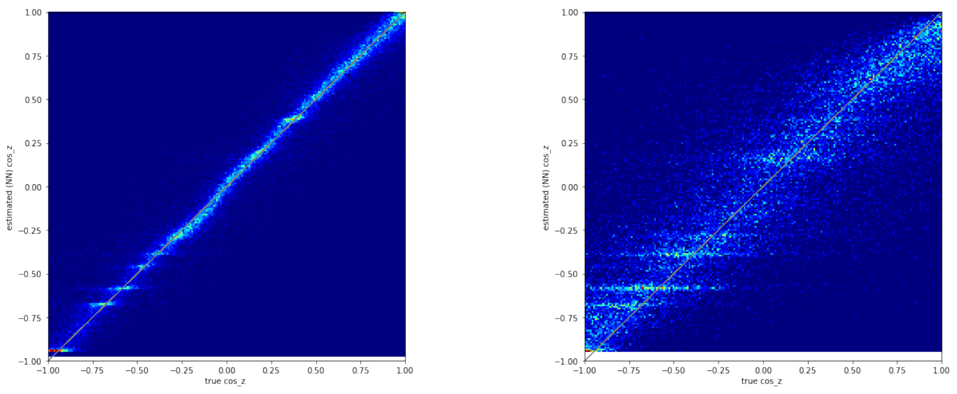


FIGURE 4.47 Regression performance of $cos(\theta_z)$ estimation for ν_μCC test events (left) and ν_eCC test events (right).

4.3 Comparison with the official reconstruction

In order to benchmark the performances of the defined Neural Network models, the results obtained have been compared, where applicable, to those obtained by the official reconstruction algorithm used in KM3NeT, namely `JGandalf`. The `JGandalf` package reconstructs neutrino events by fitting the positions and directions of the particles, assuming a *track-like* hypothesis (see Section 2.3.3). For this reason, since the algorithm is optimised to reconstruct the track signature, the comparison could be done only on the ν_μCC events. For a fair comparison, the same test events used to evaluate the performances of the Neural Network models have been used as input data for the reconstruction algorithm, and then the two estimations have been compared. The total number of ν_μCC events used is $30,469$ events (out of $51,818$ in the whole test set).

It is worth mentioning that in this comparison the Neural Network have only been used in *inference* mode, to generate estimates. In other words, no additional training of the network has been performed to carry out this comparison. Thus, the internal parameters used by the Neural Network are the same optimised after the training considering both ν_μCC and ν_eCC events.

As for the tasks, the following have been considered in the comparison:

- energy estimation comparison (4.3.1);
- $cos(\theta_z)$ estimation comparison (4.3.2);
- up-going/down-going classification comparison (4.3.3).

All the comparisons have been performed by applying two different selection criteria on the ν_μCC events. In particular, a first comparison has been done on all the selected ν_μCC events in the test set (i.e. $30,469$). Then a second comparison has been performed, by selecting the events according to the *quality cuts* reported in [6]. These cuts, optimised for the `JGandalf`-reconstructed events, are:

$$-\,lik > 60 \texttt{ AND } -log(\beta_0) > 2.8, \tag{4.1}$$

where *lik* refers to the *likelihood of the reconstructed track* (with respect to the real track), and β_0 is the error on the direction reconstruction, as reported in the `JGandalf` output.

According to this criteria, the reconstructed events have been selected, and the performances of the two algorithms (namely `JGandalf` and the Neural Network models) have been evaluated on the same selected events.

4.3.1 Energy estimation comparison

The mean squared error (MSE) and the coefficient of determination r^2 have been calculated for the two $log_{10}E$ estimations, namely Neural Network and `JGandalf`, on ν_μCC events, obtaining the results shown in table 4.4.

	Neural Network	**JGandalf**
MSE$(log_{10}E, \hat{log_{10}E})$	0.176	0.690
$\mathbf{r^2}(log_{10}E, \hat{log_{10}E})$	0.860	0.455

TABLE 4.4 Comparison of the energy estimation errors for Neural Network and `JGandalf` approaches without quality cuts.

The corresponding curves, showing the relationship between estimated and true energy values, with confidence levels at 68% and 90% highlighted, are reported in Figure 4.49 for the Neural Network and the reconstruction algorithm, respectively.

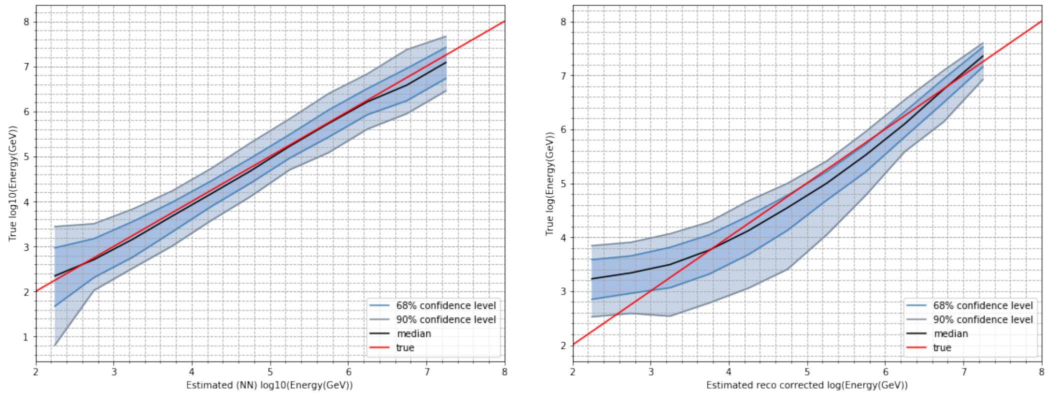


FIGURE 4.48 Performances of $log_{10}(E)$ estimation for ν_μCC events without quality cuts. Left plot: Neural Network; Right plot: `JGandalf`.

The same comparison, applying the *quality cuts* to select the events, leads to the following results. As seen in Table 4.5, the estimation error for the reconstruction algorithm drastically improves with this selection.

	Neural Network	**JGandalf**
MSE$(log_{10}E, \hat{log_{10}E})$	0.081	0.169
$\mathbf{r^2}(log_{10}E, \hat{log_{10}E})$	0.930	0.857

TABLE 4.5 Comparison of the energy estimation errors for Neural Network and `JGandalf` approaches with quality cuts applied to select events

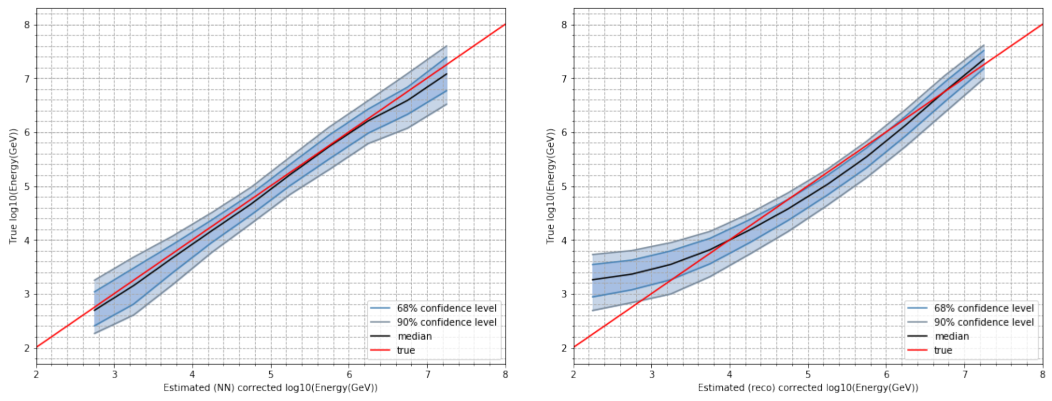


FIGURE 4.49 Performances of $log_{10}(E)$ estimation for ν_μCC events with quality cuts. Left plot: Neural Network; Right plot: `JGandalf`.

The corresponding curves of the relation between estimated and true energy values, with confidence levels at 68% and 90%, are reported in Figure 4.49 for the Neural Network and for the reconstruction algorithm. These two curves show that, while the Neural Network estimation, despite the low statistics at high energies, is rather stable (i.e. the dispersion around the median is almost constant in all the energy range), in the case of `JGandalf` the deviation is higher for low energies, whereas the estimation performances improve as the energies increase. It is also worth mentioning that on the sample used for this analysis the shape of the dependency between real energy and estimated energy for the official reconstruction is sizeably different from what reported e.g. in the KM3NeT Letter of Intent [6]. Biases in the sample have been sought but have not been found so far.

4.3.2 $cos(\theta_z)$ estimation comparison

As for the estimation of $cos(\theta_z)$, the same kind of comparisons have been performed in the two considered cases, namely with all the ν_μCC test events and by selecting the events according to the `JGandalf`-optimised quality cuts.

For the first case, the mean squared errors (MSE) on the $cos(\theta_z)$ estimation, obtained by running the Neural Network prediction on the ν_μCC test subset and the reconstruction algorithm (`JGandalf`) are:

	Neural Network	**JGandalf**
MSE$(cos(\theta_z), \widehat{cos(\theta_z)})$	0.012	0.028
$\mathbf{r^2}(cos(\theta_z), \widehat{cos(\theta_z)})$	0.960	0.904

TABLE 4.6 Comparison of the $cos(\theta_z)$ estimation errors for Neural Network and `JGandalf` approaches without quality cuts.

According to the mean error values, the Neural Network estimates seems to have better performances if compared to the reconstruction algorithm. On the other hand, more information about these results can be retrieved if the distributions of the estimated values versus the true $cos(\theta_z)$ values are considered for both approaches. Figure 4.50 shows that, while for the Neural Network almost all estimated values gather around the `true` diagonal line, values computed by the official reconstruction are *closer* to the `true` diagonal, but in the latter case, there are more spurious points distant from the true line. Thus, the mean error for `JGandalf` is higher, because of these randomly-estimated $cos(\theta_z)$ values, but the performances on the "well-estimated" values are better.

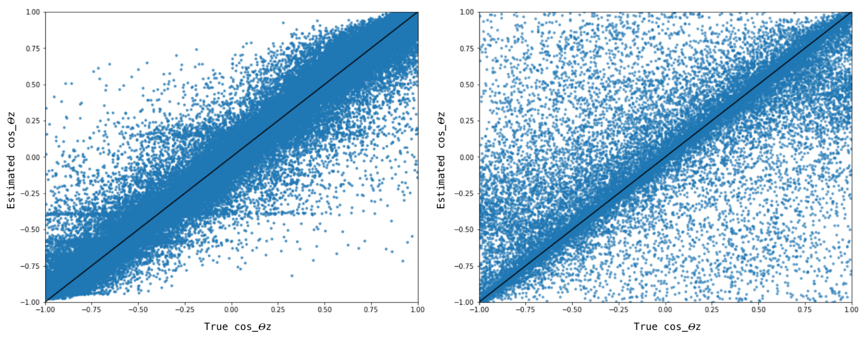


FIGURE 4.50 Scatter plots of estimated versus true $cos(\theta_z)$ values without quality cuts. Left plot: Neural Network; Right plot: `JGandalf`.

As a consequence, the corresponding confidence level curves show that for the reconstruction algorithm the 90% confidence interval is much narrower with respect to the Neural Network case, while the latter shows a more regular trend (Figure 4.51). This

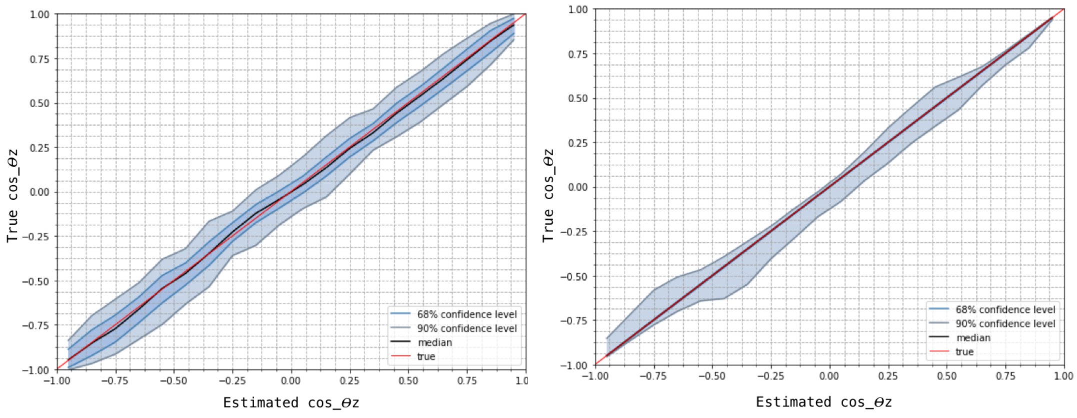


FIGURE 4.51 Performances of $cos(\theta_z)$ estimations for ν_μCC events without quality cuts. Left plot: Neural Network; Right plot: `JGandalf`.

is partially expected: the official reconstruction is using the actual DOM positions, whereas the network only gets the regularised version. Improvements on this item are expected soon.

The same calculation, performed after selecting the events according to the quality cuts, gives the results shown in Table 4.7. Here it can be noted that the performances improve for both algorithms, but the best estimation is provided by `JGandalf`.

	Neural Network	**JGandalf**
MSE$(cos(\theta_z), \hat{cos}(\theta_z))$	0.002	0.001
$\mathbf{r^2}(cos(\theta_z), \hat{cos}(\theta_z))$	0.990	0.993

TABLE 4.7 Comparison of the $cos(\theta_z)$ estimation errors for Neural Network and `JGandalf` approaches with quality cuts applied to select events.

This is also seen in the scatter plots of the estimated values as a function of the true $cos(\theta_z)$ values, shown in Figure 4.52.

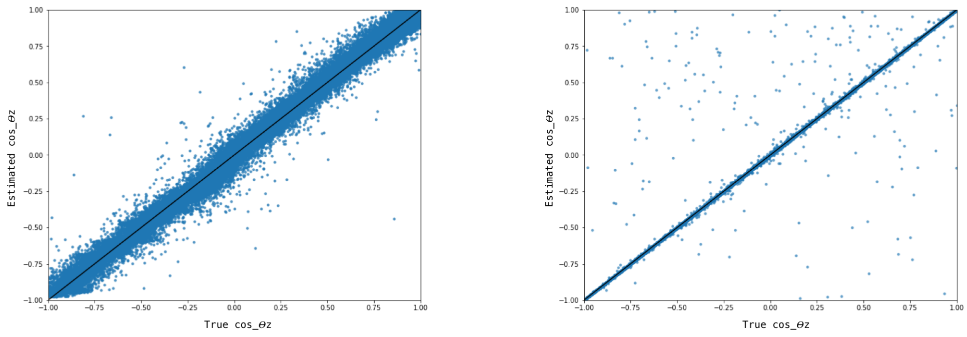


FIGURE 4.52 Scatter plots of estimated versus true $cos(\theta_z)$ values with quality cuts applied to select events. Left plot: Neural Network; Right plot: `JGandalf`.

The performances of the two different approaches, with the confidence intervals at 68% and 90%, are reported in Figure 4.53 for the Neural Networks and the reconstruction algorithm respectively. As for the `JGandalf` estimation, the confidence intervals are very narrow, indicating that most of the estimated values are very close to the target values. A *zoomed* plot, in which the intervals are visible, is shown in Figure 4.54.

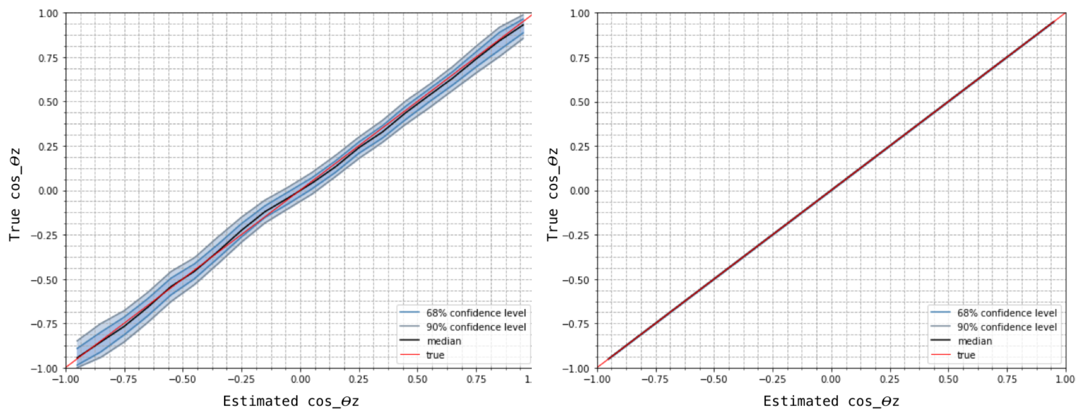


FIGURE 4.53 Performances of $cos(\theta_z)$ estimations for ν_μCC events with quality cuts applied to select events. Left plot: Neural Network; Right plot: `JGandalf`.

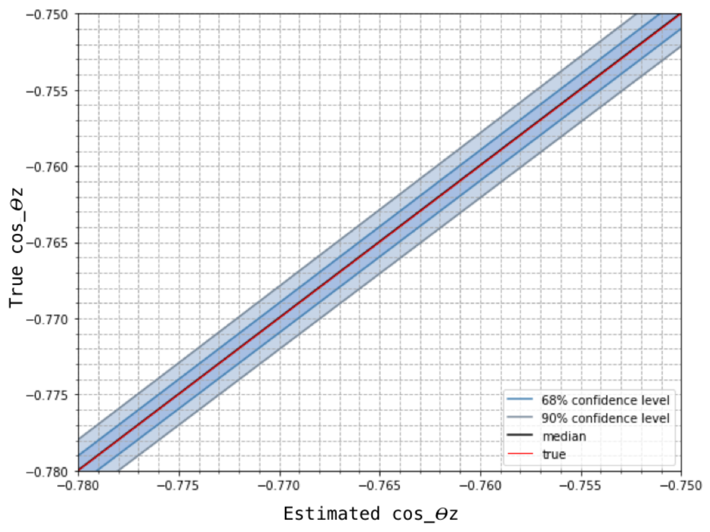


FIGURE 4.54 Reconstruction performance on $cos(\theta_z)$ estimation for ν_μCC events with quality cuts - zoom.

4.3.3 Up-going/Down-going classification comparison

The Up-going/Down-going classification, performed with the network shown in Section 4.2.1, is compared to the official reconstruction algorithm in the following way: the reconstructed values for the $cos(\theta_z)$ are used to define the reconstructed event as *up-going* or *down-going*, in the same way described for the labels definition:

$$cos(\theta_z) > 0 : up - going, \quad cos(\theta_z) \leq 0 : down - going. \tag{4.2}$$

Thus, the Accuracy is evaluated for both classifications, using only the ν_μCC test events.

	Neural Network	**JGandalf**
$\mathbf{ACC_{up/down}}$	97.5%	95.6%

TABLE 4.8 Comparison of the up-going/down-going classification accuracy for ν_μCC events for Neural Network and `JGandalf` approaches without quality cuts.

The ROC curve has been evaluated as a performance estimator for both algorithms. The comparison of the ROC curves is shown in Figures 4.55-4.57. In particular, in Figure 4.55 the whole ν_μCC test dataset is considered, whereas in Figures 4.56 and 4.57 events are split according to the energy and the distance from the detector centre, respectively. In this case the ROC curves have been calculated using the predicted classes as reference values, namely 0 or 1 for down-going and up-going, respectively. Hence, step curves are obtained: the ROC curves shown in section 4.2.1 consider the probability values yielded by the Softmax layer of Neural Networks. Since no probabilities or estimation score is provided by `JGandalf`, only the classes are used.

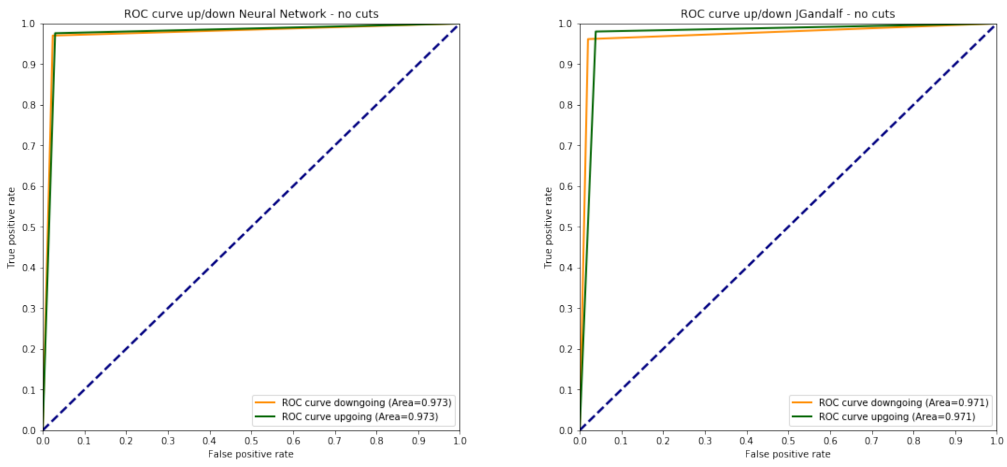


FIGURE 4.55 ROC curve comparison for up-going/down-going classification on ν_μCC events without quality cuts. Left plot: Neural Network; Right plot: `JGandalf`.

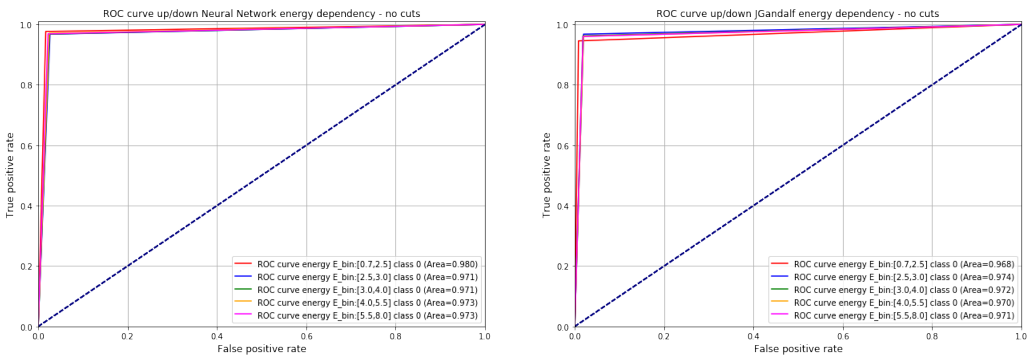


FIGURE 4.56 ROC curve comparison for up-going/down-going classification on ν_μCC events without quality cuts as a function of the simulated energy. Left plot: Neural Network; Right plot: `JGandalf`.

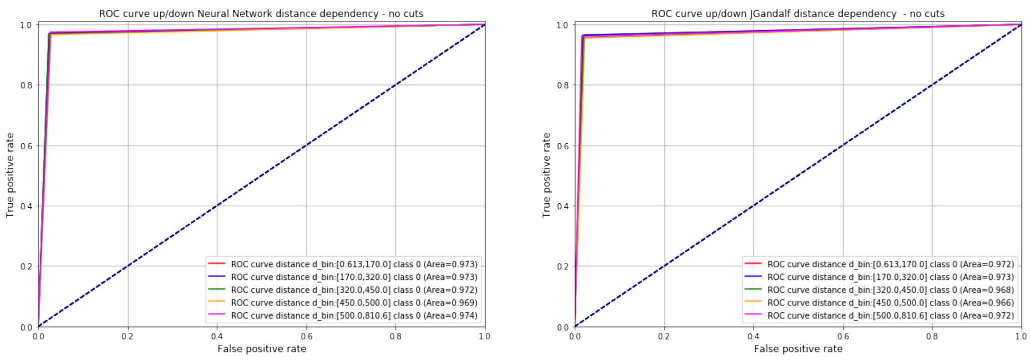


FIGURE 4.57 ROC curve comparison for up-going/down-going classification on ν_μCC events without quality cuts as a function of the distance from the detector centre. Left plot: Neural Network; Right plot: `JGandalf`.

With quality cuts better performances are reached in both cases (see Table 4.9 for the comparison of accuracies).

	Neural Network	**JGandalf**
$\mathbf{ACC_{up/down}}$	98.7%	99.8%

TABLE 4.9 Comparison of the up-going/down-going classification accuracy for ν_μCC events for Neural Network and `JGandalf` approaches with quality cuts applied to select events.

Figure 4.58- 4.60 shows the ROC curves considering the same settings described above, i.e. whole dataset, energy and distance dependency.

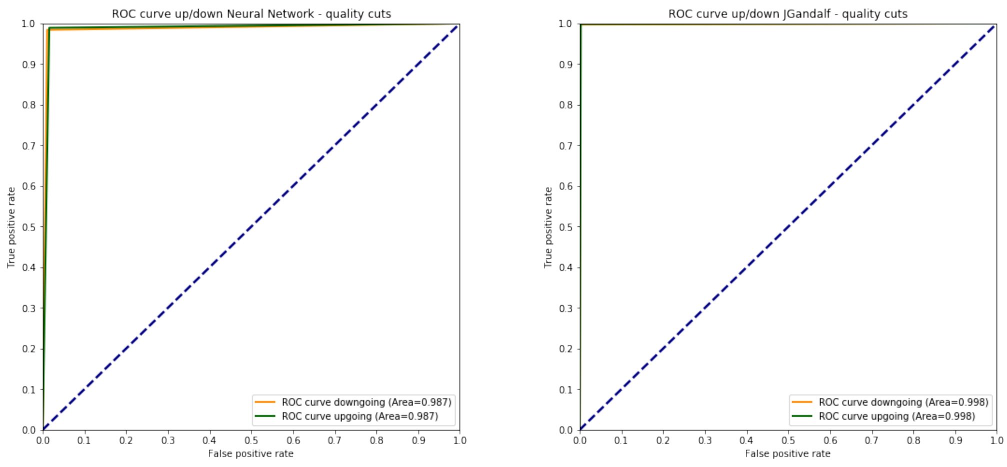


FIGURE 4.58 ROC curve comparison for up-going/down-going classification on ν_μCC events with quality cuts applied to select events. Left plot: Neural Network; Right plot: `JGandalf`.

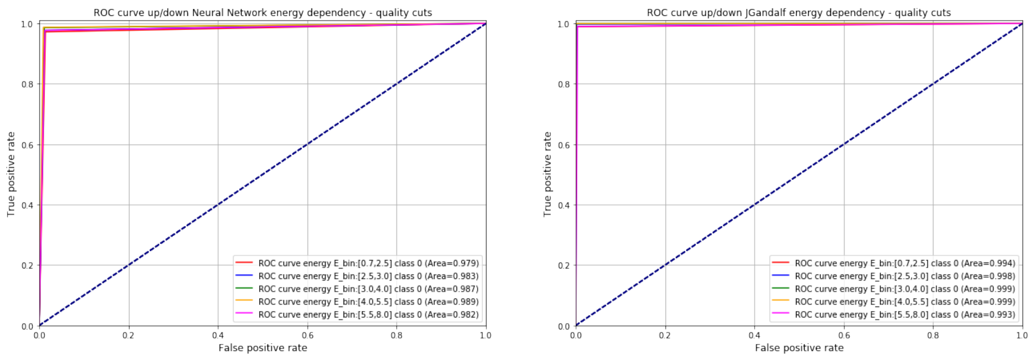


FIGURE 4.59 Performances of up-going/down-going classification for ν_μCC events with quality cuts applied to select events, as a function of the simulated energy. Left plot: Neural Network; Right plot: `JGandalf`.

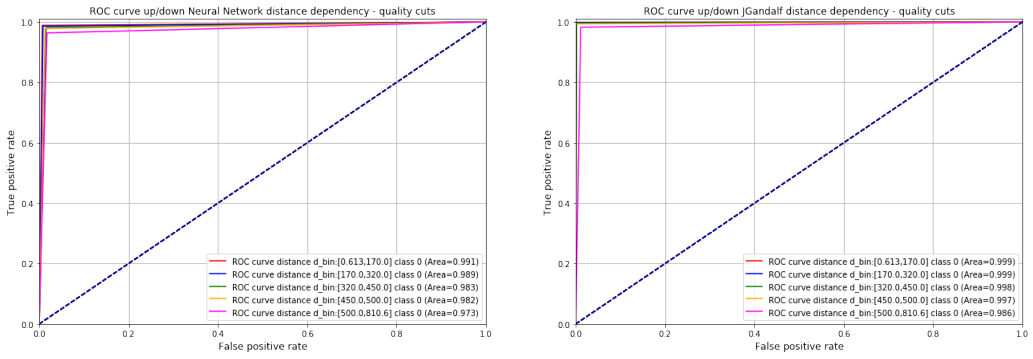


FIGURE 4.60 ROC curve comparison for up-going/down-going classification on ν_μCC events with quality cuts applied to select events, as a function of the distance from the detector centre. Left plot: Neural Network; Right plot: `JGandalf`.

4.4 Hardware settings and execution times

Unlike classical Machine Learning models, Deep Neural Networks have the advantage of leveraging the computational power provided by Graphical Processing Units (GPUs). GPUs are more efficient than CPUs in performing matrix multiplications, thanks to the large number of cores available on chip to parallelise operations. This is particularly the case of the defined Neural Network models, in which 4 to 6 million parameters (i.e. gradients and weights) have to be calculated at each learning epoch. To this aim, an `NVIDIA GEFORCE GTX-1080Ti X2` (see technical specifications in Table 4.10) has been used to perform the calculations.

Technical parameters	
Number of CUDA cores	3584
Base Clock (MHz)	1480
Boost Clock (MHz)	1582
Memory Clock	11 Gbps
Memory Configuration	11 GB
Memory Bandwidth (GB/s)	484

TABLE 4.10 Technical specifications of the GPU used to train and test the Neural Network models.

Depending on the specific model, different training times have been experienced. In particular, 3 to 4 hours were needed to perform the training for all the tested Neural Network models, considering a dataset of $207,061$ sample events. Once a Neural Network model is trained, the evaluation phase - in which the network is used only in inference mode - takes much shorter time. In fact, 2 to 4 minutes were needed to test all the presented models on the dedicated test dataset consisting of $51,818$ events. Details on the training and test times, as well as the number of parameters of each model, are shown in Table 4.11.

Model	**N. of parameters**	**Training time**	**Test time**
Up-going/Down-going class.	$6.9 \cdot 10^6$	3h 12m	4m 4s
ν_μCC/ν_eCC classification	$4.4 \cdot 10^6$	3h 58m	3m 21s
$log_{10}(E)$ estimation	$4.4 \cdot 10^6$	4h 30m	2m 27s
$cos(\theta_z)$ estimation	$5.1 \cdot 10^6$	3h 30m	2m 49s

TABLE 4.11 Train and test times of the 4 Neural Network models. The training times refers to a dataset of $207,061$ events, while the test dataset is composed by $51,818$ events.

As for the comparison of the performances with the official reconstruction algorithm (`JGandalf`), the performances reported in Table 4.11 cannot be considered because of the different experimental settings. In fact, the Neural Network models have been trained on a dataset consisting of ν_μCC and ν_eCC events, while `JGandalf` is used to process only ν_μCC events. Therefore, a different comparison must be considered. The dataset of all ν_μCC events contains $149,706$ samples, and the time required by `JGandalf` to process the whole dataset on a single CPU core is approximately 24 h[5]. The same dataset has been fed into the pre-trained Neural Network models to generate the estimations[6]. This calculations took 8 minutes to complete, thus constituting a significant boost in the performances.

[5]This time can be further reduced if multiple cores are used.

[6]Notice that no training is required.

Chapter 5

Conclusions

5.1 Summary

The presented Deep Learning applications look promising for the KM3NeT analyses. The Neural Network models have proven to be comparable to the official algorithms in terms of performances, and can already be considered as complementary to some of the current on-going analyses. In particular, the ability to perform an Up-going/Down-going classification, could be useful in the event selection/background suppression. For instance, the presented models could be used as a preliminary check to be run on the raw data before any reconstruction algorithm is applied, or act as a further test to investigate on the events discarded by the other algorithms. As for the parameters estimation, the ability to evaluate the energy and the directions of the particles is fundamental in KM3NeT, and it would be worthwhile to further investigate on the possibility to extract these (and more) information directly from raw data.

5.2 Outlook

The Neural Network-based analyses presented in the previous sections could be improved in several ways. First of all, a more refined description of the events and of the detector could be a next-order approximation worth introducing. In particular for the direction estimations, including the positions and directions of each PMT, i.e. refining the description of the detector response, as well as that of the hits, could result in a significant improvement of the performances. Nevertheless, the direction of the incoming neutrino would be better estimated if the 3 components of the direction cosine, namely $cos(\theta_x)$, $cos(\theta_x)$, $cos(\theta_z)$, were estimated simultaneously, requiring the normalisation of the overall vector. As for the particle interaction identification, the analysis could be enhanced by including all three neutrino flavours, in order to distinguish among ν_μ, ν_e and ν_τ events, or implementing a classifier able to recognise the atmospheric muon tracks and distinguish them from the signatures of neutrino events. Finally, the knowledge learnt with the previously discussed improvements could be applied to a detailed signal/background classification, which includes all the components of the two classes. A more technical improvement, on the other hand, would be possible by taking into account the irregularities in the detector, i.e. using real positions of the detector components. Furthermore, the distortions by sea currents, caused by the flexibility of the strings, could be included to make the study more realistic. Finally, inefficiencies, very common in the real usage of the detector, such as the possibility to have DOMs or PMTs not-working/switched-off, might make the technique even more robust. This is surely a promising research field with an exciting potential for further improvements.

Bibliography

[1] W. C. Haxton, R. G. Hamish Robertson, and A. M. Serenelli, "Solar Neutrinos: Status and Prospects," *Ann. Rev. Astron. Astrophys.*, vol. 51, pp. 21–61, 2013.

[2] V. F. Hess, "Über Beobachtungen der durchdringenden Strahlung bei sieben Freiballonfahrten," *Physikalische Zeitschrift*, vol. 13, pp. 1084–1091, November 1912.

[3] J. Blümer, R. Engel, and J. R. Hörandel, "Cosmic rays from the knee to the highest energies," *Progress in Particle and Nuclear Physics*, vol. 63, pp. 293–338, oct 2009.

[4] A. M. Hillas, "Cosmic Rays: Recent Progress and some Current Questions," in *Conference on Cosmology, Galaxy Formation and Astro-Particle Physics on the Pathway to the SKA Oxford, England, April 10-12, 2006*, 2006.

[5] K. Greisen, "End to the cosmic-ray spectrum?," *Physical Review Letters*, vol. 16, pp. 748–750, Apr. 1966.

[6] S. Adrian-Martinez *et al.*, "Letter of intent for KM3NeT 2.0," *J. Phys.*, vol. G43, no. 8, p. 084001, 2016.

[7] K. Mannheim, "High-energy neutrinos from extragalactic jets," *Astroparticle Physics*, vol. 3, no. 3, pp. 295 – 302, 1995.

[8] R. J. Protheroe, "High-energy neutrinos from blazars," *ASP Conf. Ser.*, vol. 121, p. 585, 1997.

[9] J. e. a. Beringer, "Review of particle physics," *Phys. Rev. D*, vol. 86, p. 010001, Jul 2012.

[10] N. Agafonova *et al.*, "Discovery of τ Neutrino Appearance in the CNGS Neutrino Beam with the OPERA Experiment," *Phys. Rev. Lett.*, vol. 115, no. 12, p. 121802, 2015.

[11] M. Markov and I. Zheleznykh, “On high energy neutrino physics in cosmic rays,” *Nuclear Physics*, vol. 27, no. 3, pp. 385 – 394, 1961.

[12] A. Roberts, “The birth of high-energy neutrino astronomy: A personal history of the dumand project,” *Rev. Mod. Phys.*, vol. 64, pp. 259–312, Jan 1992.

[13] V. A. et al, “The baikal neutrino experiment,” *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 626-627, pp. S13 – S18, 2011.

[14] E. A. et al., “The amanda neutrino telescope: principle of operation and first results,” *Astroparticle Physics*, vol. 13, no. 1, pp. 1 – 20, 2000.

[15] E. V. Bugaev, A. Misaki, V. A. Naumov, T. S. Sinegovskaya, S. I. Sinegovsky, and N. Takahashi, “Atmospheric muon flux at sea level, underground, and underwater,” *Phys. Rev. D*, vol. 58, p. 054001, Jul 1998.

[16] S. L. Glashow, “Resonant scattering of antineutrinos,” *Phys. Rev.*, vol. 118, pp. 316–317, Apr 1960.

[17] J. A. Formaggio and G. P. Zeller, “From eV to EeV: Neutrino Cross Sections Across Energy Scales,” *Rev. Mod. Phys.*, vol. 84, pp. 1307–1341, 2012.

[18] R. Gandhi, C. Quigg, M. H. Reno, and I. Sarcevic, “Ultrahigh-energy neutrino interactions,” *Astropart. Phys.*, vol. 5, pp. 81–110, 1996.

[19] A. Trovato, *Development of reconstruction algorithms for large volume neutrino telescopes and their application to the KM3NeT detector*. PhD thesis, Universitá degli Studi di Catania - Scuola Superiore di Catania, 2014.

[20] K. e. a. Olive, “Review of particle physics,” *Chin. Phys.*, vol. C38, p. 090001, 2014.

[21] M. Ackermann *et al.*, “Optical properties of deep glacial ice at the South Pole,” *J. Geophys. Res. Atmos.*, vol. 111, no. D13, p. D13203, 2006.

[22] C. D. Mobley, *Light and water: Radiative transfer in natural waters*. San Diego: Academic Press, 1994.

[23] *Proceedings, 34th International Cosmic Ray Conference (ICRC 2015)*, vol. ICRC2015, 2015.

[24] KM3NeT-Collaboration, *KM3NeT Website*.

[25] C. W. James, "GENHEN release v7r6 - KM3NeT Internal Note," 2016.

[26] G. Carminati, A. Margiotta, and M. Spurio, "Atmospheric MUons from PArametric formulas: A Fast GEnerator for neutrino telescopes (MUPAGE)," *Comput. Phys. Commun.*, vol. 179, pp. 915–923, 2008.

[27] D. J. L. Bailey, "KM3 v2r1: User Guide. ANTARES Internal note," 2002.

[28] M. de Jong et al, "The JPP software package," *www.km3net.org*.

[29] D. Real and K. Collaboration, "The electronics readout and data acquisition system of the km3net neutrino telescope node," *AIP Conference Proceedings*, vol. 1630, no. 1, pp. 102–105, 2014.

[30] K. Georgakopoulou, C. Spathis, G. Bourlis, A. Tsirigotis, A. Birbas, A. Leisos, M. Birbas, and S. E. Tzamarias, "A 100-ps Multi-Time over Threshold Data Acquisition System for Cosmic Ray Detection," 2017.

[31] B. Bakker, *Trigger studies for the Antares and KM3NeT neutrino telescopes.* MSc dissertation, Particle and Astroparticle physics - University of Amsterdam - FNWI, Institute: Nikhef - Antares/KM3NeT Collaboration, 2011.

[32] S. Eichie, *Triggerstudien zum KM3NeT-ARCA Neutrinoteleskop.* MSc dissertation, Erlangen Centre for Astroparticle Physics, 2016.

[33] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly of Applied Mathematics*, vol. 2, pp. 164 – 168, 1944.

[34] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431 – 441, 1963.

[35] P. Flach, *Machine Learning: The Art and Science of Algorithms that Make Sense of Data.* Cambridge University Press, 2012.

[36] P. Harrington, *Machine Learning in Action.* Manning Publications Company, 2012.

[37] T. M. Mitchell, *Machine Learning.* New York, NY, USA: McGraw-Hill, Inc., 1 ed., 1997.

[38] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning.* Springer, 2001.

[39] F. Chollet, *Deep Learning with Python.* Manning Publications Company, 2017.

[40] C. Bishop, *Pattern Recognition and Machine Learning.* Information Science and Statistics, Springer, 2006.

[41] J. G. Carbonell, R. S. Michalski, and T. M. Mitchell, "Machine learning: A historical and methodological analysis," *AI Magazine*, vol. 4, no. 3, pp. 69–79, 1983.

[42] J. Anderson, R. Michalski, J. Carbonell, and T. Mitchell, *Machine Learning: An Artificial Intelligence Approach.* No. v. 2 in Machine Learning: An Artificial Intelligence Approach, Morgan Kaufmann, 1986.

[43] S. Marsland, *Machine Learning: An Algorithmic Perspective.* Taylor & Francis, 2011.

[44] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction.* Adaptive Computation and Machine Learning Series, Bradford Book, 1998.

[45] R. Duda, P. Hart, and D. Stork, *Pattern classification.* Pattern Classification and Scene Analysis: Pattern Classification, Wiley, 2001.

[46] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis.* Cambridge University Press, 2004.

[47] S. Raschka, *Python Machine Learning.* Packt Publishing, 2015.

[48] W. S. McCulloch and W. Pitts, "Neurocomputing: Foundations of research," ch. A Logical Calculus of the Ideas Immanent in Nervous Activity, pp. 15–27, Cambridge, MA, USA: MIT Press, 1988.

[49] C. Van Der Malsburg, "Frank rosenblatt: Principles of neurodynamics: Perceptrons and the theory of brain mechanisms," in *Brain Theory* (G. Palm and A. Aertsen, eds.), (Berlin, Heidelberg), pp. 245–248, Springer Berlin Heidelberg, 1986.

[50] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, oct 1986.

[51] F. Rosenblatt, *The Perceptron, a Perceiving and Recognizing Automaton Project Para.* Report: Cornell Aeronautical Laboratory, Cornell Aeronautical Laboratory, 1957.

[52] S. U. S. E. Laboratories, B. Widrow, U. S. O. of Naval Research, U. S. A. S. Corps, U. S. A. Force, and U. S. Navy, *Adaptive "adaline" neuron using chemical "memistors."*. 1960.

[53] R. D. Reed and R. J. Marks, *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks.* Cambridge, MA, USA: MIT Press, 1998.

[54] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 5 2015.

[55] C. M. Bishop, *Neural Networks for Pattern Recognition.* New York, NY, USA: Oxford University Press, Inc., 1995.

[56] Y. Bengio, "Learning deep architectures for ai," *Found. Trends Mach. Learn.*, vol. 2, pp. 1–127, jan 2009.

[57] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[58] F. Chollet *et al.*, "Keras." `https://github.com/keras-team/keras`, 2015.

[59] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines.," in *ICML* (J. Fürnkranz and T. Joachims, eds.), pp. 807–814, Omnipress, 2010.

[60] Y. LeCun, K. Kavukcuoglu, and C. Farabet, *Convolutional networks and applications in vision*, pp. 253–256. 2010.

[61] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," pp. 1097–1105, 2012.

[62] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[63] A. Aurisano, A. Radovic, D. Rocco, A. Himmel, M. D. Messier, E. Niner, G. Pawloski, F. Psihas, A. Sousa, and P. Vahle, "A Convolutional Neural Network Neutrino Event Classifier," *JINST*, vol. 11, no. 09, p. P09001, 2016.

[64] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *Journal of Physiology (London)*, vol. 195, pp. 215–243, 1968.

[65] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” *ArXiv e-prints*, Mar. 2016.

[66] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” *ArXiv e-prints*, mar 2016.

[67] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[68] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, “API design for machine learning software: experiences from the scikit-learn project,” in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

[69] A. P. Bradley, “The use of the area under the roc curve in the evaluation of machine learning algorithms,” *Pattern Recognition*, vol. 30, no. 7, pp. 1145 – 1159, 1997.

[70] G. Biau and E. Scornet, “A Random Forest Guided Tour,” *ArXiv e-prints*, Nov. 2015.

[71] O. Anava and K. Y. Levy, “k*-Nearest Neighbors: From Global to Local,” *ArXiv e-prints*, Jan. 2017.

[72] B. P. Roe, H.-J. Yang, J. Zhu, Y. Liu, I. Stancu, and G. McGregor, “Boosted decision trees as an alternative to artificial neural networks for particle identification,” *Nuclear Instruments and Methods in Physics Research A*, vol. 543, pp. 577–584, May 2005.

[73] E. Drakopoulou, E. Tzamariudaki, and C. Markou, “EReNN: Energy reconstruction with neural networks.”

[74] K. Pikounis, “High energy starting muons.”

[75] A. Trovato, “MVA in point-source analysis.”

[76] R. Brun and F. Rademakers, “ROOT - An object oriented data analysis framework,” *Nuclear Instruments and Methods in Physics Research A*, vol. 389, pp. 81–86, feb 1997.

[77] S. Geißelsöder, “Shallow and deep learning applications in km3net.”

[78] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015. Software available from tensorflow.org.

[79] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, “Theano: a CPU and GPU math expression compiler,” in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, jun 2010. Oral Presentation.

[80] F. Seide and A. Agarwal, “Cntk: Microsoft’s open-source deep-learning toolkit,” in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’16, (New York, NY, USA), pp. 2135–2135, ACM, 2016.

[81] KM3NeT-Collaboration, “Km3net wiki page.”

[82] S. Maneewongvatana and D. M. Mount, “Analysis of approximate nearest neighbor searching with clustered point sets,” *eprint arXiv:cs/9901013*, Jan. 1999.

[83] S. Maneewongvatana and D. M. Mount, “It’s okay to be skinny, if your friends are fat,” in *Center for Geometric Computing 4th Annual Workshop on Computational Geometry*, 1999.

[84] E. Jones, T. Oliphant, P. Peterson, *et al.*, “SciPy: Open source scientific tools for Python,” 2001–. [Online; accessed <today>].

[85] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *ArXiv e-prints*, Sept. 2014.

[86] Image-net_team, *Image-Net*.

[87] M. Lin, Q. Chen, and S. Yan, “Network In Network,” *ArXiv e-prints*, Dec. 2013.

[88] M. D. Zeiler, “ADADELTA: An Adaptive Learning Rate Method,” *ArXiv e-prints*, Dec. 2012.