Development and commissioning of a machine learning algorithm for real time reconstruction of electromagnetic showers with a scintillating fibres tracker

Paul de Bryas
paul.debryas@epfl.ch

Supervisors:

Prof. Dr. Lesya Shchutska
Dr. Elena Graverini

January 17, 2020

CERN-THESIS-2020-405
11/02/2020

Abstract

This thesis approaches the problem of reconstructing electromagnetic showers in real time using a tracking detector interleaved with other layers serving as absorbing material. It finds immediate application in experiments such as SHiP and SND@LHC, which use calorimeters made of emulsion bricks interleaved with scintillating fibre tracking stations.

We developed a machine learning algorithm able to reconstruct the energy of an electron which showers through such a detector. The spatial location of the hits detected by the tracker planes are given as input to the algorithm. We train the algorithm using a simulated data sample with electrons energies in the $[0.3, 100]$ GeV range. We also developed a downsizing procedure which allows the algorithm to be used with tracking planes of arbitrary, larger dimensions, such as the ones used in SHiP and SND@LHC.

We achieved to use the studied detector as a sampling calorimeter, with a measured stochastic term of $a = 0.665 \pm 0.02$ GeV$^{1/2}$. We obtain a relative energy resolution of about 10% in an energy range $[40, 70]$ GeV. Even if the main calorimetry information in these experiment will be provided, at the analysis stage, by the emulsion bricks, this algorithm allows to reconstruct showers in real time using information from the tracker. As such, it will prove useful both for neutrino physics, and to spot early hints of the possible presence of dark matter.

Contents

Chapter 1

Introduction

The SHiP experiment (Search for Hidden Particles) aims at exploring the domain of very weakly interacting particles and studying the properties of tau neutrinos. It is planned for 2026, as soon as the CERN Super Proton Synchrotron (SPS) resumes operation after Long Shutdown 3. The SHiP collaboration has already submitted a detailed technical proposal [1] and it is about to submit a more detailed Comprehensive Design Report, with all the subsystems having gone through the prototyping phase.

The SHiP experiment aims at searching for particles predicted by theoretical models that complement the Standard Model (SM). Those new particles have not been observed yet. If they exist, either they are too heavy to be produced in the interactions we are able to make: in the LHC, the invariant mass of proton-proton collisions is 13 TeV, that means that the constituent quarks and gluons collide with an available centre-of-mass energy of a few TeV maximum. Or they interact too weakly, and are only produced once in a very large number of events. The first possibility can be explored by building larger accelerators which is somewhat difficult and expensive. This approach is referred as the Energy Frontier. SHiP embraces a second approach, the so-called Intensity Frontier: it can discover hidden particles which couple very lightly to SM particles, thanks to the large amount (2×10^{20}) of proton-target collisions in five years that the SPS can provide. Among those hidden particles, Heavy Neutral Leptons (HNLs) are one of the main priorities of SHiP [2]. Such particles could explain unsolved questions of the SM, such as the absence of antimatter in the Universe, the pattern of neutrino masses and oscillations and even provide a Dark Matter (DM) candidate.

The Scattering and Neutrino Detector (SND) (section 2.2.2) is a detector that will be part of the SHiP experiment. It was designed to study neutrinos of all species, which will be very copiously produced at the SHiP target, but it is also ideally suited to look for DM in a specific mass-coupling range. It is unlikely to see one of these events, but if these DM particles can scatter on ordinary matter (it is expected that they do, with very small cross section), then if one puts enough matter along the path of a DM particle, one or more such event(s) will eventually be observed. This new detector needs a very precise tracker, with a resolution of few μm, as we need to be able to distinguish between various neutrino flavours and various kinds of interactions. A good timing resolution would be necessary in order to use a time-of-flight technique to distinguish DM from neutrino scattering. To achieve such a resolution, the tracker will be a combination of emulsion bricks and layers of scintillating fibres trackers (SciFi) such as the one adopted

for the LHCb experiment upgrade [3].

This thesis focusses on the development of an algorithm to reconstruct electromagnetic (EM) showers in the SND. Emulsions are known for their very good spatial resolution, but have the downside that they need to be taken out from the detector and developed in order to extract information, and therefore they are unsuitable for real time analysis. Being able to reconstruct events in which a particle has initiated an EM shower in the SND is however of extreme importance both for neutrino physics and to spot early hints of possible presence of DM. The SciFi layers can be used for this purpose. Unfortunately, ordinary tracking techniques such as those in use at the LHC experiments are not suitable for the reconstruction of EM showers, characterised by too high occupancy for these techniques.

This is where Machine Learning (ML) can be of help. ML refers to algorithms that computers use to learn from training data in order to become able to make predictions on future, unseen data. Once the algorithm is well trained, which can be done before the operation phase, it is then faster to make predictions on new data. ML has been known for half a century but it is only since recently that this field of information science has been largely focussed on and developed. Those kinds of algorithms needs a high computing power and capability to store and process a huge amount of data. Moreover the development of increasingly realistic simulations, that emulate what happens during particle collisions and how particles interact with detectors, help for the training phase of ML algorithms. For all those reasons, ML has begun to hit its stride in the world of particle physics. To give an idea of how important those algorithms have become, at LHCb, 70% of all data retained are classified by ML algorithms and all charged-particle tracks are vetted by neural networks [4].

The goal of this Master Thesis project is to develop a ML algorithm able to recover some features of a particle which went through a tracker. More specifically, we want to find the energy of an electron which has crossed a detector which looks like the SND that will be used for SHiP. The electron will produce an EM shower, and the particles forming the shower will cross the different tracker planes. The hits detected in those planes are used as input by the algorithm.

The first part of the report is a general description of the proposed SHiP experiment, insisting on the SND design. In the second part, we will present the ML tools that we used for this project. We will focus on deep learning programs that use Artificial Neural Network, and more specifically Convolutional Neural Network. Finally in the third part, we will present the algorithm and the results we got, among which the resolution with which the initial energy of the showering electron can be reconstructed, and also different tricks we used to improve the performance of the algorithm.

Chapter 2

The proposed SHiP experiment

2.1 Motivations for the SHiP experiment

Despite describing most particle physics phenomena with great precision, the Standard Model (SM) is an incomplete theory. There are fundamental physical phenomena observed in nature that the SM does not explain. The following list enumerates some of them:

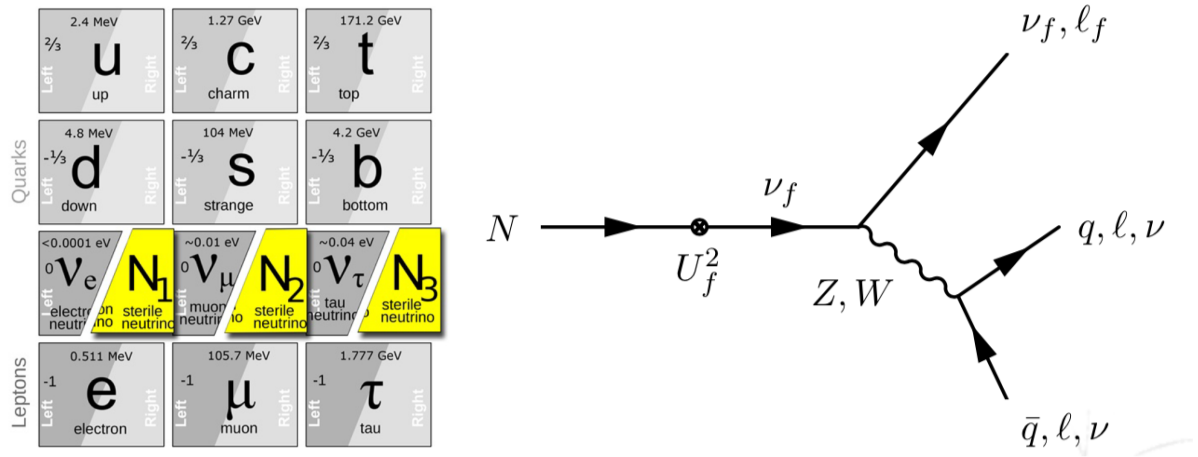
1. Dark matter (DM): cosmological observations tell us that most of the matter present in the universe is not composed of visible particles, but rather of DM particles which interact very weakly with the SM particles. The SM does not supply any fundamental particles that are good DM candidates.

2. Neutrino masses/oscillation: experimental observations have proven the phenomenon of neutrino oscillation, and thus of non-zero neutrino masses, which is in contradiction with the SM.

3. Matter–antimatter asymmetry: SM predicts that matter and antimatter should have been created in equal amounts at the big bang. However, measurements show that matter is much more abundant than antimatter in the universe, and the sources of CP symmetry violation in the SM are not enough to account for this imbalance.

In addition to the above experimental facts, the SM also has some theoretical problems: what is the origin of the 19 fundamental constants, why does quantum chromodynamics (QCD) seems to preserve CP-symmetry (strong CP problem)? The SM is far from being the grand unified theory physicists yearn to discover.

Theoretical physicists have proposed various forms of Beyond the Standard Model (BSM) new physic theories. This is the reason why the SHiP experiment was proposed: searching for hidden particles that are predicted by those new theories, that could radically transform our understanding of nature. Those hidden particles SHiP seeks for are very weakly interacting long lived particles, predicted by a very large number of BSM models which are capable of accommodating DM, neutrino oscillations, and the origin of the full baryon asymmetry in the Universe. Hereinafter, we briefly present the different type of particles predicted by BSM models SHiP can look for.



(a) (b)

Figure 2.1: *(a) νMSM adds three new fundamental fermions to the SM, right-handed Majorana HNLs: N_1, N_2 and N_3. (b) Decay of a HNL through mixing with a SM neutrino [6].*

2.1.1 Heavy Neutral Leptons

Discovering right-handed Majorana neutrinos, also called Heavy Neutral Leptons (HNL), represents the primary goal of SHiP. Those particles are predicted, among many theories, by the "Neutrino Minimal Standard Model" (νMSM) [5]. In the SM, neutrinos are massless and only left-handed, but in the νMSM, three right-handed counterparts N_1, N_2 and N_3 are added to the particle content of the SM (see Figure 2.1a). N_1 is a DM candidate, because it could have a lifetime greater than the age of the Universe, and a mass of few keV. The other two HNLs, in the few GeVs mass range, could explain the baryon-antibaryon asymmetry of the Universe and the origin of neutrino masses. We could observe HNLs through their decay to a variety of SM final states (see Figure 2.1b) through the emission of a $W^{\pm}$ or Z^0 boson.

2.1.2 Dark Sector

HNLs are part of a category of particles that belong to the so-called hidden or Dark Sector, which are unobserved particles with weak couplings to the SM and with masses below the electroweak scale. Some theories suppose the existence of Dark Sector particles with a mass range between the MeV and tens of GeV, associated to neutral mediators [7] which interact both with the dark sector and with the SM. These mediators, which must be singlets under the SM gauge symmetry, can lead to couplings of feebly-interacting particles to the SM through various operators, which we call "portals" (Figure 2.2).

Which means that, through the portals, it is possible to discover dark sector particles by looking for their interaction with SM particles. For example, HNLs belong to the so-called neutrino portal. In the same way, dark photons (vector portal, i.e. a massive gauge boson mixing with the SM photon), dark scalars (Higgs portal) and Axion Like Particles (ALPs) may well be accessible by SHiP experiment if they have a mass in the

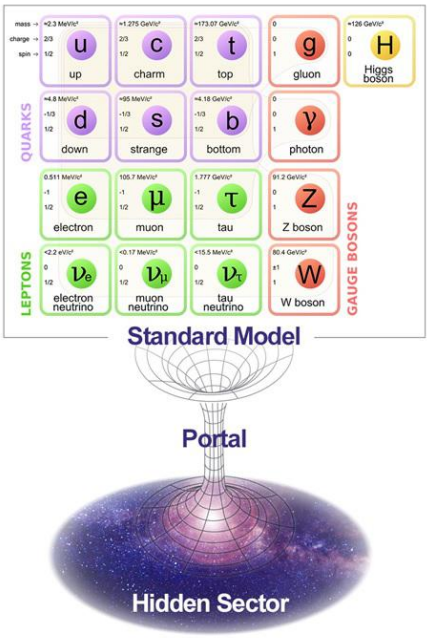


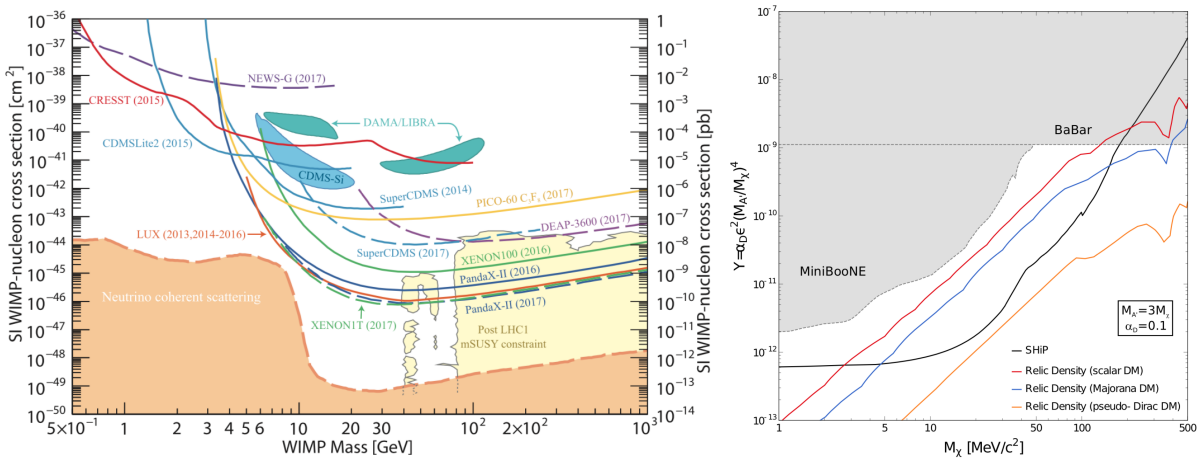
Figure 2.2: *Portals mix or connect the dark sector particles with the SM particles [7].*

MeV-GeV range and can thus be produced in the decay of c and b mesons produced at the SHiP target.

One of the most interesting portals is the vector portal, which foresees a massive gauge boson γ' charged under a new gauge symmetry $U'(1)$, and coupled to the SM by kinetic mixing with the SM photon. Some additional stable particles χ may exist, that are charged under $U'(1)$ but do not directly interact with the SM (they do only through their coupling to γ'). It follows that these are ideal DM candidates. Dark photons may be produced at the SHiP target by Bremsstrahlung from the beam protons or their constituents, or in the decays of secondary mesons. After being produced, γ' may decay to pairs of DM particles (which have a chance to produce a signal in the SND) or to SM particles (and would thus be detected in the Decay Spectrometer, described in section 2.2).

2.1.3 Supersymmetric Goldstone bosons

Looking for light hidden particles predicted by the Supersymmetry (SUSY) theory is also an objective for SHiP. In this theory, each particle would have an associated "superpartner", that can be detected only at very high energy. Solution to the gauge hierarchy problem implies the SM superpartners are at or below the TeV energy scale. However, supersymmetric Goldstone bosons can exist as well in a low-mass range accessible at SHiP: we call them sgoldstinos [12]. The couplings of these new particles are expected to be very weak. Since these particles are expected to be long lived and they could decay to SM particles, they might be discovered at SHiP.



(a) (b)

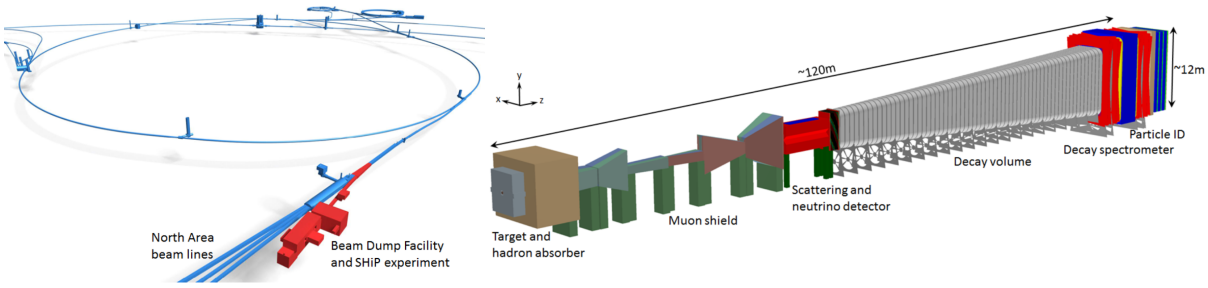
Figure 2.3: *(a) WIMP cross sections (normalized to a single nucleon) for spin-independent coupling versus mass. The DAMA/LIBRA [8], and CDMS-Si enclosed areas are regions of interest from possible signal events, to be confirmed by other experiments. For context, the black contour shows a scan of the parameter space of 4 typical SUSY models, CMSSM, NUHM1, NUHM2, pMSSM10 [9], which integrates constraints set by ATLAS Run 1 [10]. (b) Expected sensitivity of the SHiP detector in the parameter space of LDM [11].*

2.1.4 Light dark matter

Light Dark Matter (LDM) are Weakly Interacting Massive Particles (WIMPs) with masses in or below the GeV range. WIMPs are hypothetical new elementary particles, produced thermally in the early Universe, and are possible candidates to constitute DM. Figure 2.3a shows the exclusion limits for WIMPs in a parameter space defined by mass (along the x axis) and coupling (i.e. cross-section for DM-matter scattering, along the y axis). As one can see, the low-mass range is much less constrained, which is what makes experiments like SHiP or LDMX [13] very interesting. The reason why only large masses have been explored is that these experiments use huge volumes of dense material, and search for recoiling nuclei, which would be a sign that DM scattered on them. This technique needs a heavy WIMP to induce a detectable nuclear recoil. There are different models and constraints for LDM candidates [14] and SHiP is sensitive to some of them in a certain mass range (see figure 2.3b). The vector portal described in section 2.1.2 is one such example. SHiP can detect LDM scattering on the material of the SND detector, looking for a single EM shower induced by a scattered electron, when there is no other activity in the event. Techniques to achieve this purpose have been developed in this master's thesis work.

2.2 The SHiP detectors

SHiP is a proposed general-purpose experiment to be installed at a new beam dump facility [16] at the Super Proton Synchrotron (SPS), whose setup is shown in Figure 2.4a. It all began in 2013, when 13 researchers have proposed a new fixed-target experiment



(a) (b)

Figure 2.4: *(a) Overview of the proposed SPS Beam Dump Facility [15]. (b) Overview of the SHiP experiment as implemented in FairShip [15].*

at the CERN SPS accelerator to search for HNLs [2]. Few years after that proposal, a collaboration of more than 300 researchers have submitted a detailed technical proposal for an experiment named SHiP: Search for Hidden Particles [1]. A progress report was released in 2018 [15], and the collaboration is about to submit an even more detailed comprehensive design reports, with all the subsystems designed and gone through the prototyping phase (e.g. test beams). The final approval decision is expected for 2020. The goal of SHiP is to reconstruct the decay vertex of very weakly interacting long lived particles produced from in beam-target interactions. Figure 2.4b gives an overview of the arrangement of the different SHiP detectors and devices as implemented in FairShip (the software framework adopted by the SHiP collaboration). The SHiP detector is composed of two complementary apparatuses: the Scattering and Neutrino Detector (SND), and the Decay Spectrometer (DS). Due to the fact that the beam dump will produce a large amount of neutrinos of all species, the SND will allow, among the rest, the first observation of the tau antineutrino $\bar{\nu}_{tau}$. Such detector will also be ideally suited to seek for LDM through scattering signatures from recoil of electrons or nuclei. The Scintillating Fibre tracker object of this thesis is part of the SND, where it is interleaved with emulsions bricks.

The SPS will deliver a total of 4.0×10^{19} protons on target (PoT) per year during five years, for a total of 2×10^{20} interactions, with a proton beam energy of around 400 GeV. The centre-of-mass energy will be around 27 GeV. The proton beam is projected onto the target, and, to maximise the production of charm and beauty hadrons, the target has been chosen to be made of a heavy material. The target is followed by a hadron stopper to shield away all SM particles except neutrinos, and by a magnetic muon shield to prevent residual muons from leaking into the detector area. Neutrinos produced at the target will be studied with the SND. Additional long-lived weakly interacting particles can be detected with the DS.

2.2.1 Decay Spectrometer

The hidden sector decay spectrometer is situated downstreams of the SND and is composed of the decay volume, the decay spectrometer, and a system for particle identification (PID). It aims at measuring the decays of HS particles into SM particles, by reconstructing their decay in a 50 m long decay volume maintained at a pressure of 1 mbar in order to

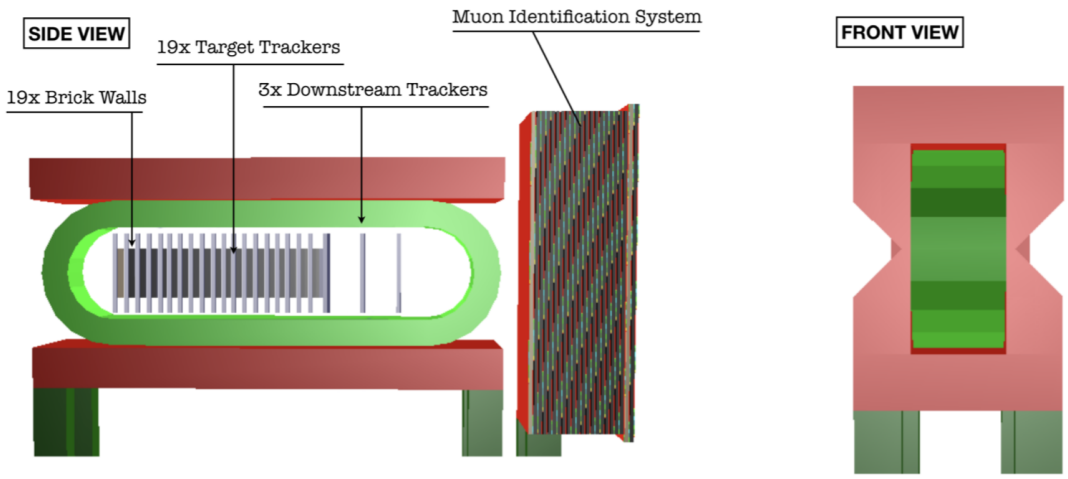


Figure 2.5: *Schematic layout of the Scattering and Neutrino Detector [15].*

eliminate the background from neutrino-air interactions. The decay volume is followed by a large spectrometer that provides a measurement of the momentum of the decay products. In addition, the tracking planes (straw tracker detector) upstream of the magnet allow to reconstruct the particle tracks and the decay vertex where they originated. PID is provided by the ECAL for electrons, photons and mesons, and by the muon system for muons. It is essential to have a low background and a precise particle ID in order to discriminate between the very wide range of HS models. Combinatorial background is reduced to a negligible level using an accurate timing detector. Particles coming into the decay volume from the outside (e.g scattering products from deviated muons) are vetoed with 30 cm of liquid scintillator surrounding the whole decay volume. Residual backgrounds are correctly identified and separated using offline data selection criteria.

2.2.2 Scattering and Neutrino Detector

General Overview

The goal of the Scattering and Neutrino Detector (SND) is to measure neutrino scattering cross sections, and in particular to measure the characteristics and cross-sections of the tau neutrino and antineutrino and to quantify charm production in neutrino scattering. Such a detector, for its characteristics, is also ideally suited to search for LDM scattering. The tau neutrino ν_τ is part of the third generation of leptons and is the second most recent particle of the SM to have been discovered [17]. The BDF facility will be constructed in such a way that it will produce a lot of neutrinos. This gives the opportunity to discover the tau antineutrino $\bar{\nu}_\tau$ and deepen our knowledge on the tau neutrino properties. Production of ν_τ is accompanied by that of τ leptons, so SHiP would also allow to search for the Lepton Flavour Violating decay ($\tau^- \to \mu^-\mu^+\mu^-$ and its charge conjugate) with high sensitivity. A dedicated experiment named TauFV has been proposed [18].

Figure 2.5 gives a schematic layout of the SND detector. It can be decomposed into two parts. The first part is the emulsion spectrometer, located inside a 7 m long magnet (green coil on the figure), which provides a 1.2 T horizontal magnetic field. The second

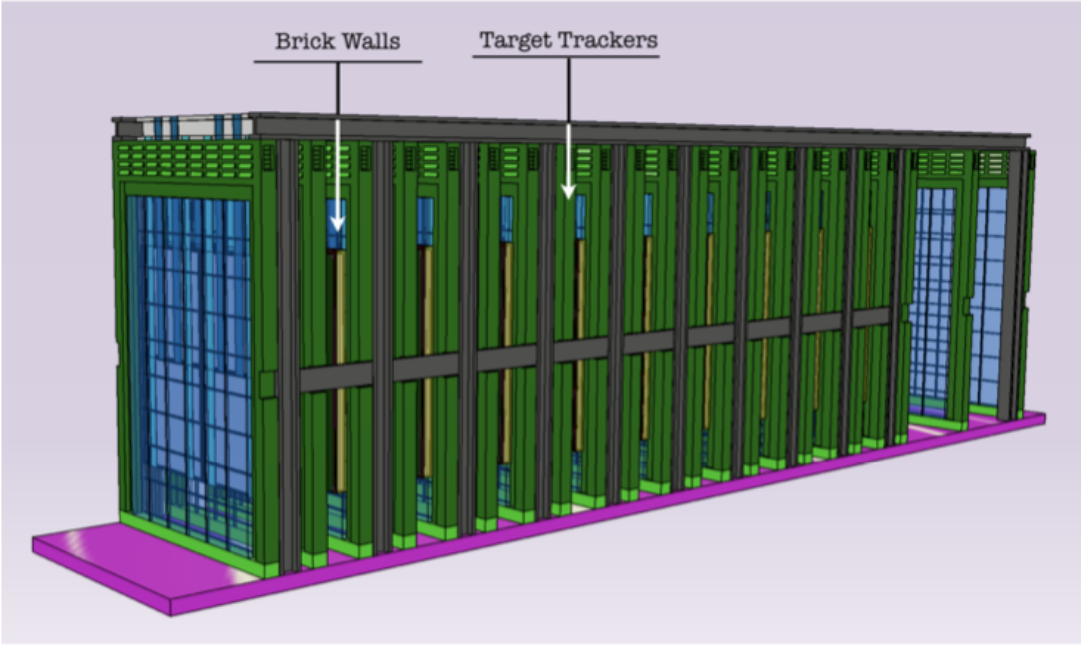


Figure 2.6: *Layout of the Emulsion Target and Target Tracker and closeup view of one emulsion brick wall of four cells, each containing an Emulsion Cloud Chamber (ECC) and a Compact Emulsion Spectrometer (CES) [15].*

part is the muon identification system, located downstream.

The emulsion spectrometer (Figure 2.6 on the left) is a hybrid detector which first part consists of an Emulsion Target interleaved with Target Tracker (TT) planes, and which second part is made of three Downstream Trackers. It was designed to distinguish between ν_μ, ν_e and ν_τ: ν_μ are identified by the presence of activity in the muon identification system, ν_e are distinguished by electron shower identification in the brick, and ν_τ by the disentanglement of τ production and decay vertices. It can also distinguish between ν_τ and $\bar{\nu_\tau}$ by performing the electric charge measurement of the τ decay products. The τ decay channels investigated are the electron, muon and hadron channels ($\tau^- \to l^- \bar{\nu}_l \nu_\tau$, $\tau^- \to h^- \nu_\tau$, $\tau^- \to h^- \pi^0 \nu_\tau$).

The emulsion target is made of "brick walls" with Emulsion Cloud Chambers (ECC) made of lead plates (required to maximise the number of neutrino interactions) interleaved with nuclear emulsion films, and followed by a Compact Emulsion Spectrometer (CES) (Figure 2.6 on the right). It has very high spatial resolution and it is well suited for the measurement of charged particle momenta and for electron identification. The momentum is measured by the multiple coulomb scattering in the lead plates and electrons can be identified by detecting the EM showers, but the emulsions have to be mechanically taken out and passed through a microscope in order to see what happened in there. To make up for this problem, the brick walls are interleaved with planes of TT, (i.e. the SciFi tracker) to allows tracking and EM shower reconstruction in real time, even if with less accuracy. The TT will also provide the time stamp of the event. A good timing resolution would be necessary in order to use a time-of-flight technique to distinguish DM from neutrino scattering. Finally, the three Downstream Trackers, separated by 50 cm air gaps, have

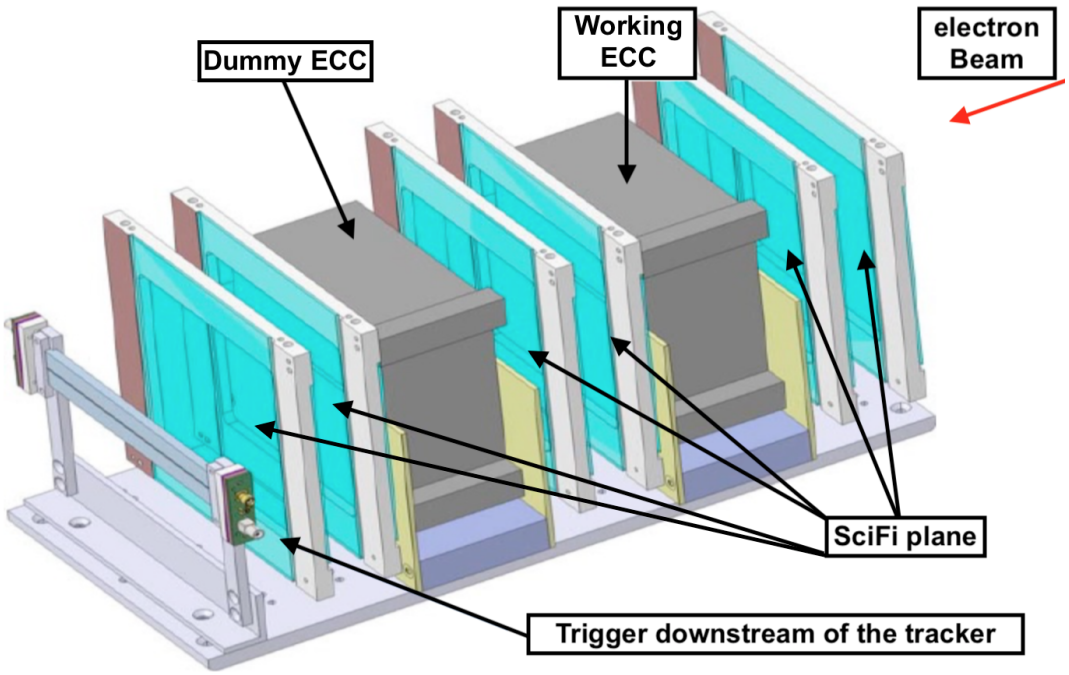


Figure 2.7: *Early stage prototype of the emulsion spectrometer, composed of one working and one dummy ECC detector, in combination with 6 SciFi planes [19].*

the role of measuring the charge and momentum of muons exiting the target region.

The muon identification system is a two meter long system made of a sequence of iron filters and Resistive Plate Chambers (RPC) planes. It is tailored to measure the charge and momentum of muons above 10 GeV/c. The system also has the role of tagging neutrino interactions in its material which could lead to long-lived neutral SM particles such as K_L entering the DS and mimicking the signal from a hidden particle decay.

Research and development for the SND

As for all the other detector parts, the SND had to pass through a research and development phase before being designed for implementation into the SHiP experiment. The objective of the SHiP collaboration is to have a fully prototyped design of the final version of the SND before 2024 [20], in order to have the time to build it and install it by the end of 2027. In 2019, a prototype of the emulsion spectrometer part of the SND (see figure 2.7) has been built by a collaboration of six research teams (among which there was an EPFL team), in order to test the capacity of such a detector in identifying EM showers and measure their energy. It was tested at the DESY test beam facility.

2.3 The FairShip software

The software framework adopted by SHiP is based on FairRoot [21] and it is called FairShip. The FairRoot framework is an object oriented simulation, reconstruction and data analysis framework. It includes core services for particle interaction, detector simulation and offline analysis. It is based on the ROOT system [22], which is a modular scientific

software toolkit. It provides all the functionalities needed to deal with data processing, statistical analysis, visualisation and storage. It is mainly written in C++ but integrated with other languages such as Python. The dependencies of FairShip are tracked and installed using alibuild, the build manager used by the ALICE experiment [23]. Git is used as version control system to track the changes of the FairShip software package over time. The framework is accessible at `https://github.com/ShipSoft/FairShip/`.

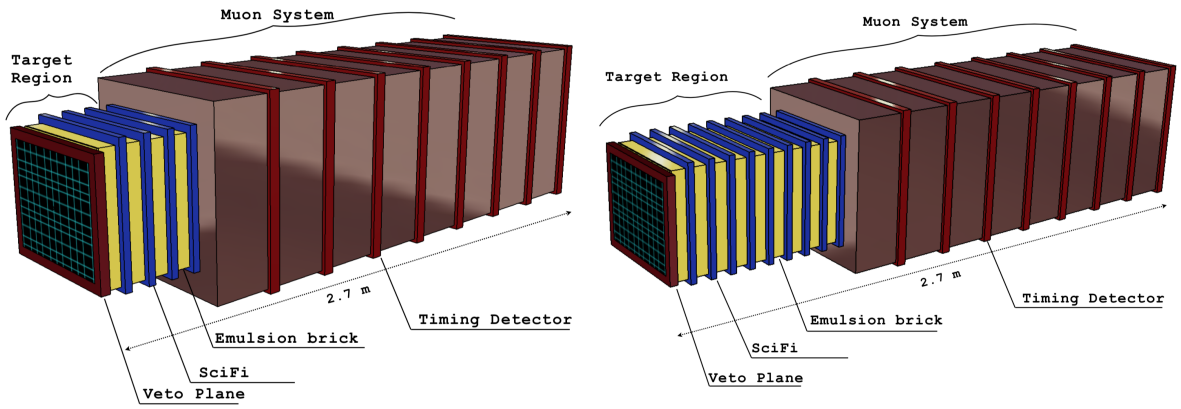
The FairShip code allows one to describe the detector geometry, implement detector classes, and perform physics simulations. Module shapes, detectors and other material are defined with the ROOT TGeo classes; the active and passive materials of the detectors are implemented in Geant4 [24], which simulates detector response; track reconstruction relies on GENFIT [25]. Events are produced using the Pythia8 [26] generators for the proton-target collision, Pythia6 [27] generators for inelastic muon scattering, and Genie [28] for SM neutrino interactions.

2.4 The SND@LHC proposal

The SND@LHC is a very recently proposed experiment (originally named XSEN) aimed at installing an SND-like detector in a side cavern of the LHC in order to study energetic neutrinos produced at one of the LHC interaction points [29, 30]. This experiment will have the unique possibility of studying neutrinos in the TeV range produced in large quantity (including the ν_τ flavour). Just like the SHiP experiment, SND@LHC will make it possible to look for LDM scattering signatures as well. Figure 2.9 shows the expected sensitivity.

After an investigation of the level of background at the different available locations, it has been decided to place the detector in a decommissioned LEP injection tunnel next to the ATLAS interaction point (LHC-IP1). The detector will look in the direction of the beam, linearly extrapolated from the position of IP1 (which is the direction neutral particles produced in the forward direction at IP1 would take). It will therefore benefit from the large amount of neutrinos generated with the pp interaction, from W and b,c decays. In order to adjust the settings of the prototype detector, a pilot run with a 2.4 m long detector should take place in 2021, and a the full run with a 2.7 m long detector should start in 2022. Figure 2.8a and 2.8b show respectively the design of these two detectors.

The SND@LHC runs will also clearly serve as test bench for the SND detector of the SHiP experiment.



(a) (b)

Figure 2.8: *Overview of the pilot run (a) and the full run (b) detector of the "SND@LHC" experiment [19].*

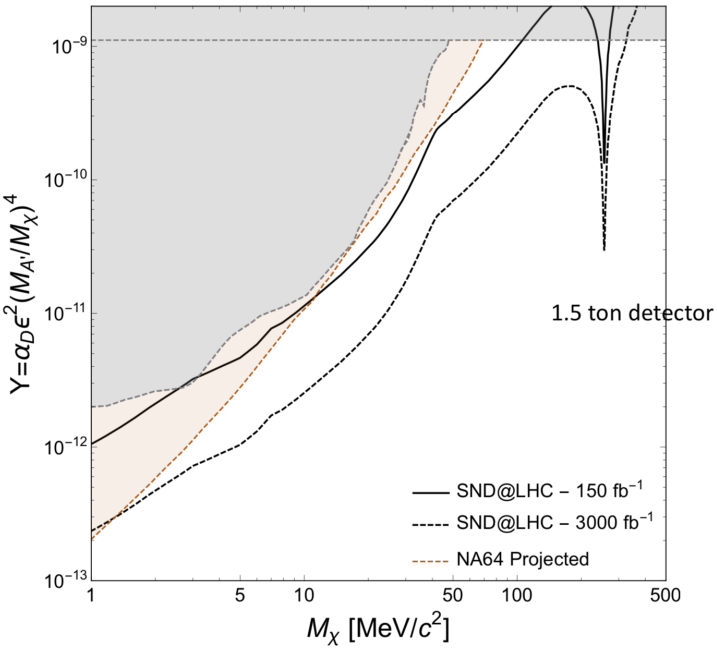


Figure 2.9: *Expected sensitivity of the SND@LHC detector in the parameter space of LDM [31].*

Chapter 3

Deep Learning for particle physics

Deep learning is a field of Machine Learning (ML) algorithms, largely used nowadays, that use an artificial neural network for learning. Artificial neural networks (ANNs) are loosely inspired by the working principle of neurons in the human brain. First we will explain the very basic principles of a deep learning program, then we will talk about convolutional neural network (CNN), which is a kind of ANN suited for deep learning algorithms that use images as input.

3.1 Basic principle of a deep learning program

As for ML, a deep learning algorithm is used to perform a specific task without having been originally specifically designed to do this task, relying on patterns and inferences of previous experiences instead. What makes the difference between deep learning and other ML algorithms is that the inner structure of a deep learning algorithm is inspired by neurons and connections between them in the human brain, that is why it is called an artificial neural network. An ANN is based on a collection of connected units called artificial neurons. A neuron can transmit a signal to other neurons, like synapses in the brain. Neurons are organised in layers, and signals travel from the first (input) to the last (output) layer, possibly after traversing inner layers multiple times. We will first talk about the ANN architecture and relations between layers, then how the algorithms actually learn, and finally how to check how accurate the algorithm is.

3.1.1 Artificial neural network architecture

To get a better representation of what an ANN is concretely, the best is to explain it with the help of a simple example. Figure 3.1 represents a very simple neural network. On the left, one can see the "input layer" composed of n parameters (I_1, I_2, ... , I_n) where n is the number of features the neural network uses to make its predictions. For example if the goal of the algorithm is to predict if it will rain or not in the next 12 h, the first parameter I_1 will be the temperature, I_2 the pressure, I_3 the humidity measurement at 8 am. On the right, there is the "output layer" which size k is the number of predictions one wants to make. To stick with our example, it will be a single number (k=1) O_1 between 0 and 1, which will represent the probability that it will rain in the next 12 h. Between the input and the output layer, there are the "hidden layers". Each of the m

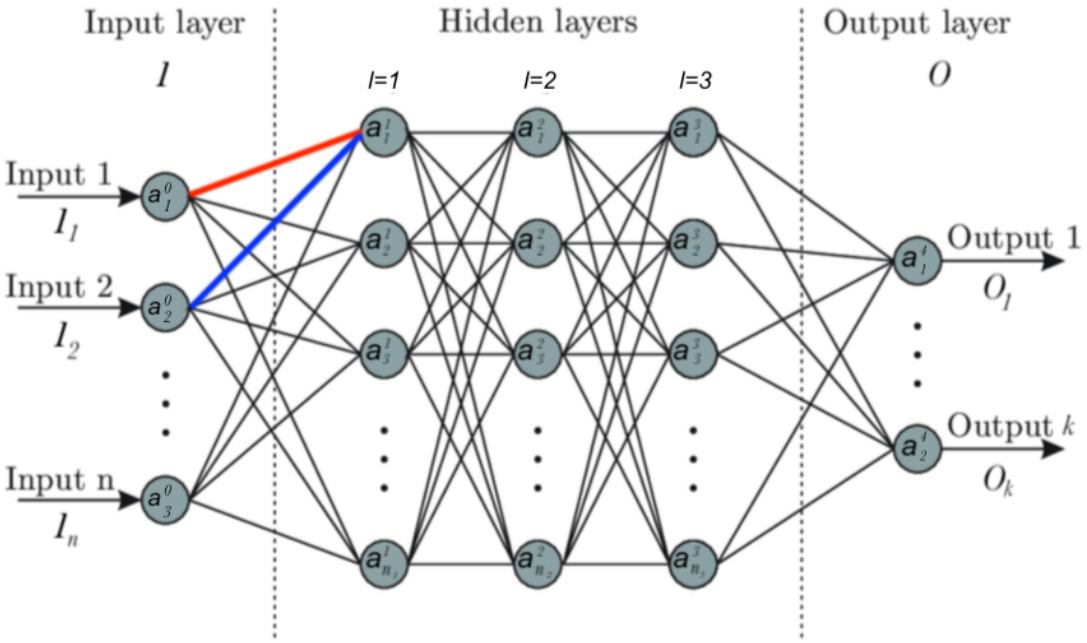


Figure 3.1: *Basic ANN structure [32]*

layers (here $m = 3$) have n_l parameters, where $l \in [0, m]$ denote the index of the layer. The parameters of the layer, called neurons, are denoted a_j^l, with $j \in [0, n_l]$ the neuron index of the layer l. a_j^l is a certain function of the parameters of the previous layer: $a_j^l = a_j^l(a_1^{l-1}, ..., a_{n_{l-1}}^{l-1})$.

This function can take many different forms, but it usually looks like this:

$$a_j^l = \sigma(\sum_{k=1}^{n_{l-1}} w_{jk}^l a_k^{l-1} + b_j^l), \tag{3.1}$$

where w_{jk}^l is a number, called the weight of the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer. The w_{jk}^l weights are represented as connections in figure 3.1. b_j^l is a number that we call "bias", associated to each neuron, and $\sigma(x)$ an activation function (we will see later what forms can $\sigma(x)$ take). Figure 3.2 gives a schematic representation of the a_j^l function, which characterise a neuron in an ANN.

To rewrite this expression in a matrix form, which will be much more convenient, we define a weight matrix W^l of size $n_l \times n_{l-1}$ for each layer. The component of the j^{th} row and k^{th} column of the weight matrix W^l is the number w_{jk}^l. Equation 3.1 can then be rewritten as:

$$\vec{a}^l = \vec{\sigma}(W^l \vec{a}^{l-1} + \vec{b}^l) = \vec{\sigma}(\vec{z}^l), \tag{3.2}$$

with $\vec{z}^l = W^l \vec{a}^{l-1} + \vec{b}^l$ the weighted input to the neurons in layer l. $\vec{b}^l$ and $\vec{a}^l$ are the bias and neuron vector, which j^{th} component are b_j^l and a_j^l respectively. $\vec{\sigma}(\vec{x})$ is simply a vector whose i^{th} component is $\sigma(x_i)$, where x_i is the i^{th} component of the vector $\vec{x}$.

Hereinafter, we will have a closer look at the role of the weights, the bias and the activation function.

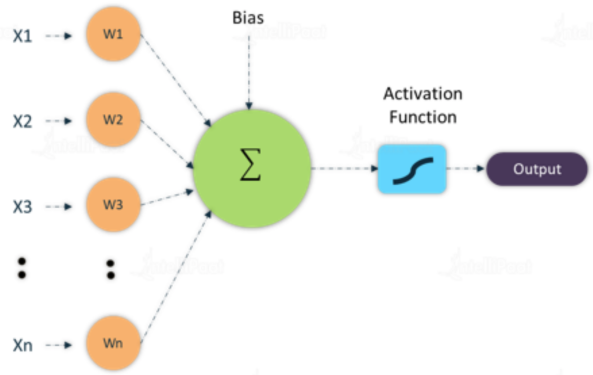


Figure 3.2: *Schematic representation of a neuron in an ANN [32].*

1. **Weights**: One way to represent an ANN is graphically, with the neurons and the connections between them, called edges, as in figure 3.1. When you give an input signal (I_1, I_2, ... , I_n) to the ANN, it will propagate through the neurons through the edges until the output layer. A weight increases or decreases the strength of the signal passing through an edge. For example, if w_{11}^1 is huge and w_{12}^1 is very small (corresponding respectively to the red and blue edge in figure 3.1), it means that the first neuron of the first layer a_1^1 is strongly correlated with the first input I_1, but not as strongly with I_2;

2. **Bias**: while weight increases or decreases the strength of the signal passing through an edge, bias is used to delay the triggering of this signal. Indeed, with the absence of bias, the model will train over points passing through the origin only, which is not in accordance with a real-world scenario. With the introduction of a bias, the model will become more flexible;

3. **Activation function**: The role of the activation function is to "compress" the value of the neuron in a more restricted range to avoid dealing with too high numbers. An example of a very known activation function called sigmoid is $\sigma(x) = \frac{1}{1+e^{-x}}$ and it compresses x into $]0, 1[$.

We have shown a very simple ANN architecture, but you could find very complex ones. The $\sigma(x)$ function and a_j^l can takes different forms. Moreover, some neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Those non-linear functions are very effective for the algorithm precision because it avoids that the output is simply a linear response of the input (not often the case in reality). Layers inter-dependency can also be changed, i.e. parameters of the a_j^l function are not necessarily the neurons of the previous layer a_j^{l-1}. Here, the weights and the bias are the parameters of the ANN, which means that they are the only variables that affect the output. A correct tuning of those parameters is essential for the algorithm to be accurate in its predictions. The tuning of weights and bias is done during the training phase.

3.1.2 Training the algorithm

Once we have built the architecture of our ANN, we enter a training process, whose goal is to find which weights and bias allow to get the best precision for the algorithm predictions.

For this part, we need a large data set in which there is a collection of inputs as various as possible, associated with the correct output, i.e. the one we expect our algorithm to predict. If we stick to the example presented in the previous section of an algorithm that predicts if it will rain or not in the next 12 h given the temperature, pressure and humidity at 8 am, we need to collect those information every day at 8 am, and see at the end of the day if it has rained or not, during one year: the corresponding output will be 1 or 0. To sum up, for each measurement x, where x is an integer that goes from 1 to t (t is the total number of measurements: for this example, $t = 365$), an input vector $\overrightarrow{I}(x)$ (here of size 3) is associated with its correct output vector $\overrightarrow{O}_{true}(x)$ (here of size 1). Once t is large enough, we can start training our algorithm.

Initialisation and cost function

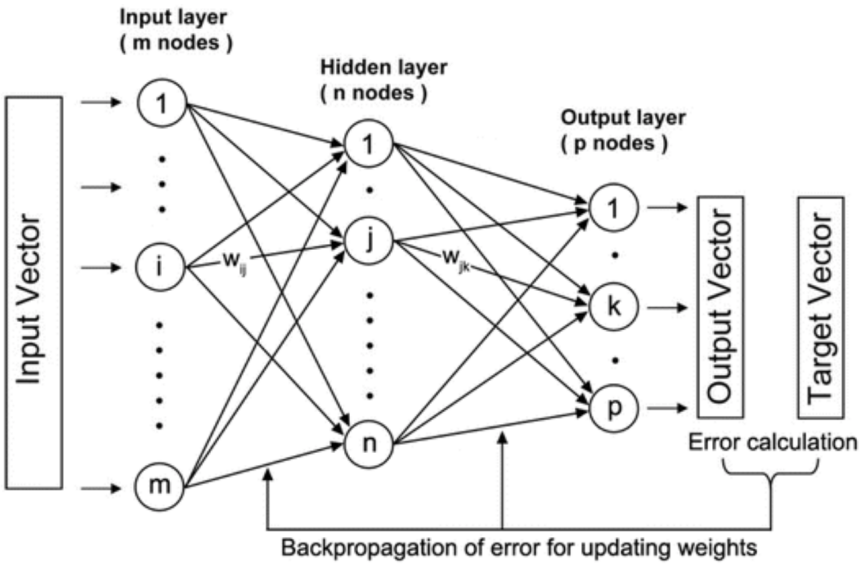
The first thing to do is to initialise the weights and bias of the network randomly. Then, using the data set, one computes, giving each of the $\overrightarrow{I}(x)$ inputs, the associated outputs predicted by the algorithm $\overrightarrow{O}_{pred}(\overrightarrow{I}(x)) = \overrightarrow{O}_{pred}(x)$. As we have set random weights and bias, the predicted outputs will be very different compared to the true ones, and we want to quantify how large this difference is for each x. This job is done by the so-called partial cost function C_x. The total cost function C measures the accuracy of the algorithm to reconstruct the true value taking into account all x. We will see that we will need to compute the partial derivatives $\frac{\delta C}{\delta w_{ij}^l}$ and $\frac{\delta C}{\delta b_j^l}$ of the total cost function. In order to compute those derivatives, C must meet two criteria: C must be an average over the cost functions for individual training samples C_x, and it must be written as a function of the outputs $\overrightarrow{O}_{pred}(x)$, for all x. One straightforward form C can take can be the sum-of-squares error:

$$C = \frac{1}{2t} \sum_{x}^{t} ||\overrightarrow{O}_{true}(x) - \overrightarrow{O}_{pred}(x)||^2,$$

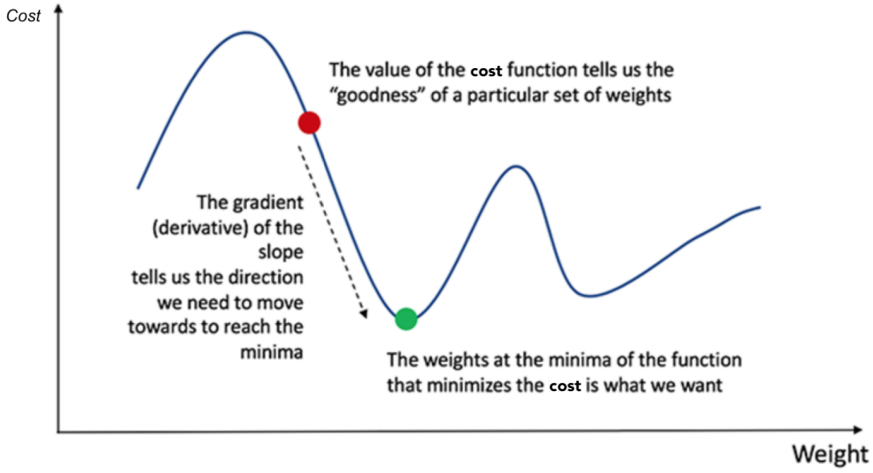
which meets the criteria cited above. One may say C also depends on the desired output $\overrightarrow{O}_{true}(x)$, but as the input training sample x is fixed, the desired output is also a constant.

Back-propagation and gradient descent step

The next step is to update the weights and bias in order to minimise the total cost function. This consists in to the well known variations calculus problem, which is a field of mathematical analysis that uses variations (small changes in functionals) to find maxima and minima of functionals [34]. The total cost function C is function of the weights and bias, and one can compute the derivative of C with respect to them. Because those derivatives are often very complex to compute, we use a graph to represent C rather than the standard formula notation. We call such a graph "computational graph" (see this



(a)



(b)

Figure 3.3: *(a) Schematic view of a three-layer neural network. Each input value is passed through the neural network. The output value is compared with the desired target output, an error is computed and this error is propagated backwards through the network to each node. (b) Explanation graph of the gradient descent. For convenience, this figure illustrates the gradient descent step for a cost function with only one weight as parameter [33].*

paper from Christopher Olah [35]): it is a way to represent a mathematical function in the language of graph theory. Using a computational graph, we can easily compute partial derivatives at every neuron, and then apply the chain rule to compute the derivative of the partial cost function C_x with respect to any weights or bias: $\frac{\delta C_x}{\delta w_{ij}^l}$ and $\frac{\delta C_x}{\delta b_j^l}$. We are not going to dwelve more deeply in the way to find those gradients because machine learning libraries, such as Keras, take care of this for us. Once we have those derivatives, we can then recover $\frac{\delta C}{\delta w_{ij}^l}$ and $\frac{\delta C}{\delta b_j^l}$ by taking the mean of $\frac{\delta C_x}{\delta w_{ij}^l}$ and $\frac{\delta C_x}{\delta b_j^l}$ over all x. Then, we can use this gradient to update the weights and biases in a way to minimise the total cost function (see figure 3.3b). This step is known as a gradient descent step and the following equations give the updated weight $(w_{ij}^l)'$ and bias $(b_j^l)'$:

$$\begin{aligned} (w_{ij}^l)' &= w_{ij}^l - \epsilon \frac{\delta C}{\delta w_{ij}^l} \\ (b_j^l)' &= b_j^l - \epsilon \frac{\delta C}{\delta b_j^l} \end{aligned} \tag{3.3}$$

where ϵ is the learning rate, which determines the step size at each iteration.

We call this kind of step "back-propagation", because the error computed at first is spread upstream the algorithm structure. Figure 3.3a gives a schematic view of how this back-propagation step works. For a standard gradient descent algorithm, a training epoch ends after a gradient descent step has been computed, i.e. when weights and bias are updated. Then one can repeat this operation: starting from the same data set inputs $\overrightarrow{I}(x)$ (for all x), one can compute outputs predicted by the newly updated network $\overrightarrow{O}_{pred}$, and perform another gradient descent step, etc. The idea is to repeat this operation until $C(w_{ij}^l, b_j^l)$ has reached a minimum.

Mini-Batch Gradient Descent

In practice, it takes a very long time for computers to add up the influence of every x for each gradient descent step (t is generally a very large number). What is commonly done instead, if parallel processing is available, is to shuffle the index of the data set and divide it into a number n_{mb} of mini-batches composed of $\lfloor \frac{t}{n_{mb}} \rfloor$ measurement each. Then, at each training epoch, one computes a gradient descent step for each mini-batch inputs $\overrightarrow{I}(x)$, so that, at the end of the epoch, the influence of every x has been taken into account. For each step, it will not be the gradient of the total cost function (which depend on all the training data), so it will not be the most efficient step to minimise C, but each mini-batch will provide a good enough approximation. It will speed up training drastically, as it will compute n_{mb} gradient descent steps per training epoch instead of one. However, one needs to make sure that the mini-batches are large enough to be sufficiently representative of all x. One can start another training epoch by shuffling again the index of the data set and repeat a mini-batch gradient descent.

Figure 3.4 provides an illustration of the differences between a standard gradient descent algorithm and a Mini-Batch Gradient Descent algorithm, for the minimisation of a two parameters function. One can see in the right-hand side figure that weights do not follow a straight line directly to the minimum but stray around the optimum path. One can think this is a bad feature of the mini-batch gradient algorithm, but in the contrary,

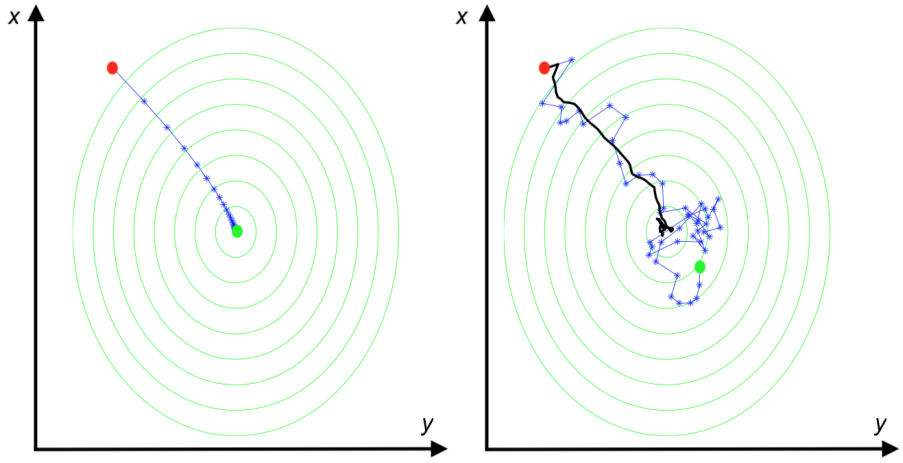


Figure 3.4: *An illustration of the gradient descent algorithm (left) and the Mini-Batch Gradient Descent algorithm (right). The function to be minimized is f(x,y)=* $1.25(x+6)^2+(y-8)^2$*. The green lines corresponds to some isolines of the function* $f(x,y)$*. For the Mini-Batch Gradient Descent, the black line depicts the averaged value of* w *during one training epoch [36].*

it is exactly what we want. Indeed, one can see on figure 3.3b, that if you start from a random point in the weight axis, you will not end up necessarily in the minimum of the cost function but maybe in a local minimum if you choose the standard gradient descent algorithm. For example, if the weight was initialised at a higher value, the red dot (starting point of the cost function, with random parameters) could have been on the right of the second hill formed by the curve, and the green dot (ending point of the cost function, with optimised parameters) would have ended up in a local minimum of the function, rather than in the global minimum. With the mini-batch gradient descent, weights and biases will not follow the optimal path to the closest local minimum, and will partially deviate from the optimal trajectory, allowing them to probe the hyperspace of the cost function. Which means that, in figure 3.3b, it would have been able to get out of a local minimum to fall into a deeper one.

3.1.3 Cross-validation with holdout method

After a certain number n_{epoch} of training epochs, the weights and bias of the network will be such that the cost function will reach a minimum from which it will not be able to come out. At this time, we want to check if the network has learned correctly the task that we asked it to do. How can we be sure of that? The first idea that comes in mind would be to simply pick a random x, take $\overrightarrow{I}(x)$ as an input of the network and compare $\overrightarrow{O}_{pred}(\overrightarrow{I}(x))$ with $\overrightarrow{O}_{true}(x)$. This would be a very bad idea, because it would not provide more information than the cost function does. Moreover, it is consequential that the network will have correct predictions for data on which it has been trained (a comparison algorithm would have better efficiency). What we really want is to verify the robustness of the network on unseen data.

The idea is to split the original data set into a training data set, on which the network

is trained, and a validation data set, to check the accuracy of the network in recovering the predicted outputs. The validation data set can also be used to know when to stop the training epoch iteration. Indeed, we can check that any increase in accuracy over the training data set (reduction of the cost function) actually yields an increase in accuracy over the validation data set. If the cost function continues to decrease, but the accuracy over the validation data set stays the same or decreases, then one should stop the training.

Deep learning algorithms have parameters (weights and bias) that are tuned during the training phase, but they also have so-called "hyper-parameters", which will not be tuned during the learning phase. We already saw some of them, such as the learning rate, the mini-batch size, the number of layers and number of neurons for each layer, etc. But there are many others. Hyper-parameters can affect the performances of the algorithm and can be optimised. Even if we have some keys to help us setting correct hyper-parameters before starting the learning phase, the exact fine tuning of them can still improve the performance of our algorithm. To do so, we train several times the exact same network with only one hyper-parameter that slightly changes. Then, we use the validation data set on each network to determine which hyper-parameter value is better. We can start again with another hyper-parameter, until all of them are correctly tuned.

Because we have tuned the hyper-parameters based on the validation set, a correlation appears between the validation set and the way we evaluate the accuracy of our algorithm, so we need another data set to evaluate the real performances of the final network with the optimal hyper-parameters. To sum up, a short description of those different data set is given here:

1. **Training data set**: used to adjust weights and bias of the ANN;
2. **Validation data set**: used to optimise the hyper-parameters and the number of training epoch iterations.
3. **Testing data set**: used only for testing the final solution in order to confirm the predictive power of the network.

Care should be taken, however, that if any of the data sets are not suitable to represent the original data, then the results from the testing data set could be biased. So what is done in practice is to do a random permutation over the index x of the original data set, and to split those data into three different data sets. The random permutation allows to have sub data set the most representative of the original one. We can take for example 80% of the original data set for the training (the most data-greedy part), 10% for validation and 10% for testing. One can easily understand now the importance of a very large initial data set for deep learning algorithms.

3.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a type of ANN, designed specifically to analyse images. CNNs employ a mathematical operation called convolution (linear operation) in at least one of their layers, in place of a generic matrix multiplication (equation 3.2) used in classical ANNs.

(a) (b)

Figure 3.5: *(a) Input image (Robin, picture taken by Chris Heald) (b) Visualization of the activation maps extracted from the first convolutional layer in the VGG16 Model [37].*

In this section, we will present first the general idea behind CNNs, and then we will present an example of inner architecture that will introduce the different type of layers we can find in CNNs.

3.2.1 The idea behind CNNs

Until now, we saw examples of algorithms which use a vector of numbers as an input, where all numbers are spatially independent of each other. If we go back to the example presented in section 3.1.1 (predicting if it will rain or not in the next 12 h), temperature, humidity and pressure are the inputs of the algorithm. Their place of entry (I_1, I_2, ..) can be interchanged without changing the accuracy of the algorithm, as long as the same ordering is kept for every x.

What happen if now we would like to use our algorithm to tell wether a figure represents dog or a cat. A very simple task for a human being, but a very complex one for a computer. If we want to keep the ANN's architecture of figure 3.1, a solution would be to "flatten" the image into a vector, where every input I_i corresponds to the value of one of the pixels of the image. For the time being, let us consider images that are all in black and white and have a dimension of $n_x \times n_y$ pixels, so that each pixel is a number between zero and one, corresponding to its shade of gray.

The first thing we need, then, is a large data set of images of dogs and cats, each of them associated with a label which value is either 0 or 1, 0 meaning it is a cat and 1 if it is a dog. Fortunately, we can find on internet such a database, e.g. "CIFAR-10", where we can find a data set of 12000 labelled images ($n_x = n_y = 32$) of either a dog or a cat (labelled, meaning that a correct output is associated with each image). The input layer of our ANN would be a vector of $n_x \times n_y = 1024$ numbers between 0 and 1, and the output layer a single number p, also between 0 and 1. After the training process, we want p to be the probability that the input image is a dog and $(1 - p)$ to be the probability that it is a cat.

A problem that comes with such an implementation is that it is computationally heavy. Indeed, the larger the input vector, the more abundant the weights and biases, and so

the computational time required to calculate the gradient descent step. For example, if we have only three layers of 30 neurons each, a quick calculation shows that for such an input image (32×32 B&W pixels), there are 32641 weights and biases that need to be tuned at each step. If for each training epoch we need to process 10000 images, it can take a very long time. Moreover, a 32×32 B&W image is not really the kind of data we have nowadays. We use to deal with RGB images (3 times more input parameters) of, usually, at least 500×500 pixels, which means that such a simple ANN architecture would have 22.5 millions of weights and biases. This leads to a very known problem in deep learning algorithms, called overfitting. Overfitting occurs when a network is unable to learn effectively: accuracy over the validation data set stays the same or decreases. It seems like that the fewer parameters are required, the less likely the network will overfit.

Moreover, classical ANNs do not take the spatial structure of the data into account. So we need to build an ANN architecture that is compatible with images and does not have too many parameters. From there the idea of CNNs was born. They are analogous to traditional ANNs in that they are comprised of weights and biases that self-optimise through learning. The output layer will be used to compute the cost function, and all that we saw in the previous section for traditional ANNs (training, cross-validation) will still apply. The only difference is the inner architecture and the different types of layers we can find.

We can find in CNNs convolutional layers: they are layers composed of "activation maps", which are 2D images (analog to the neurons of traditional ANNs). Each pixel of an activation map is connected to only a small xy region of the images in the previous layer. We can imagine an activation map as the superposition of all the 2D images of the previous layer, passed through a "filter". Each filter's goal is to represent a certain pattern of the previous images, and each filter must recognise a different pattern. For example, CNNs can be used for classifying images of animal. Figure 3.5 gives an idea of what those filters do once the CNN is already trained. Figure 3.5a shows the input image (RGB image of a bird) and figure 3.5b shows the activation maps extracted from the first convolutional layer. Each of those activation maps are like the RGB input image passed through different filters. One can see for example the activation map on top left corner: the corresponding filter helps at the recognition of the edge of the animal, the activation map just on the right of this one aims at discriminating the sky from the rest of the image. This shows the very early stage of the recognition process, but after passing through different layers, it will recognise more and more complex and abstract features, until it can classify this image as a bird.

3.2.2 Architecture and layers

As for the ANNs, the best way to understand what a CNN is, is with the help of a simple example. Figure 3.6 represents the structure of a very simple CNN, but it will be sufficient to explain all the different type of layers one can find in a CNN. We will explain all the steps from the left to the right.

First, the dimension of the input, which has to be the same for all images of the data set, is N_x, N_y and N_z. For example, $N_z = 3$ if the input is an RGB image, corresponding respectively to the red, green and blue 1D projections of the RGB image.

The first layer on the left is the convolutional layer (Conv hereinafter). It is a set

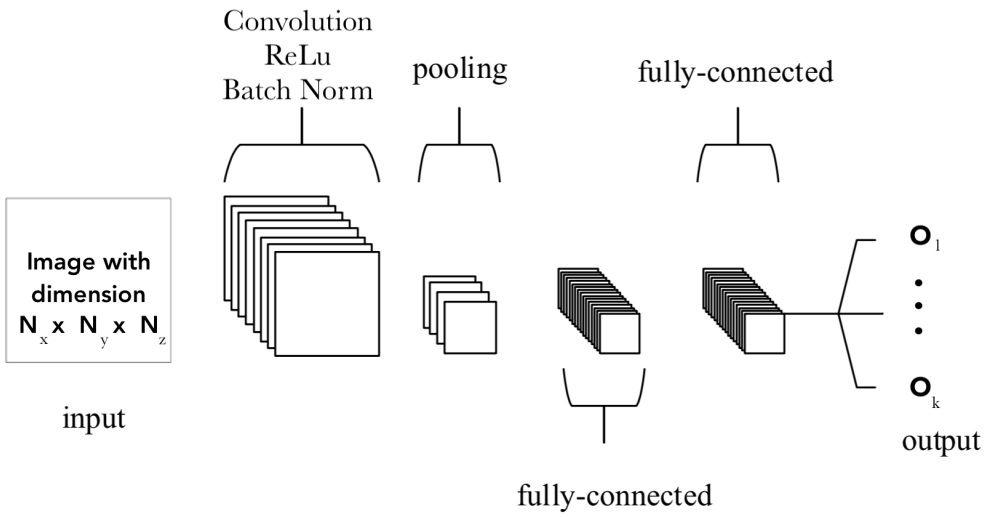


Figure 3.6: *A simple CNN architecture, comprised of just five layers [38].*

of n_c "activation maps" (8 in figure 3.6). Each of them is a convolution product of a filter and of the input image. Inside those n_c filters the weights and biases of the network are found. From Conv, we can derive other useful layers, such Coordinate Convolutional layer (CoordConv hereafter).

The Conv is combined with a Rectified Linear Unit layer (ReLU hereinafter) and a batch normalisation layer (BatchNorm hereinafter). Both functions are applied individually to each pixel of the activation map. ReLU is a non-linear function and BatchNorm has the same role as the activation function (see section 3.1.1). BatchNorm layers also have parameters that are optimised during the learning process.

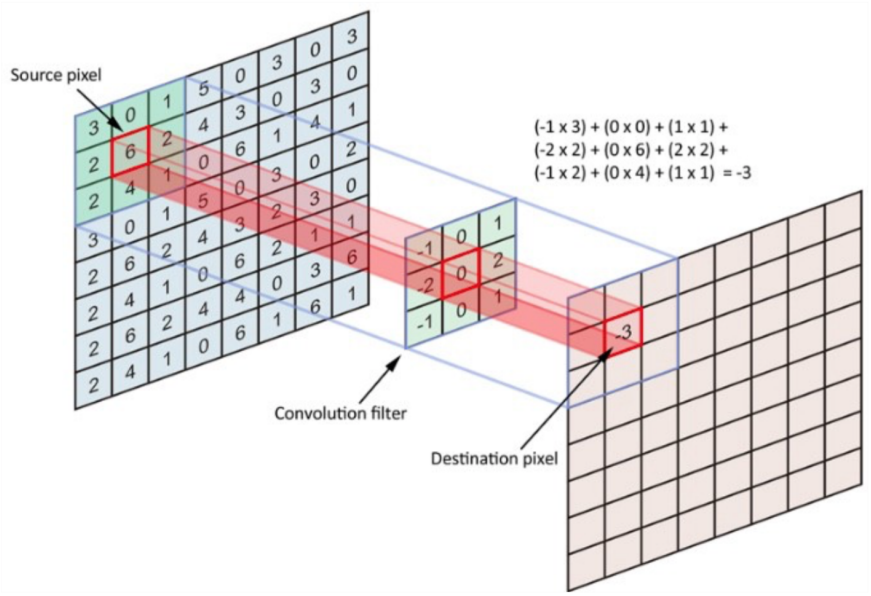
Then, we have a pooling layer (Pool hereinafter), that will simply perform a downsampling along the x, y dimensions of the activation maps of the previous layer, by applying a function that takes as argument a set of pixels close to each other (in the x, y space) and returns a single pixel.

Afterwards, we have two fully connected layers. They act exactly as standard ANN layers as shown in figure 3.1, taking all the pixels from the last layer as an input, and giving as output the number k of predictions wanted.

Hereinafter, we explain concretely how each of those layers is obtained from the previous one. All layers cited further will be used for the CNN developed for this thesis project.

Convolutional layer

A Conv is a set of D_z "activation maps", which are 2D images of $D_x \times D_y$ pixels, each of them obtained with a different filter F_i^l $(i = 1, ..., D_z)$ associated to the layer l. Each filter is a set of multiple weights and only one bias. One pixel of the activation maps in layer l can be written as $A^l_{x,y,z}$, with $x \in [1, D_x]$, $y \in [1, D_y]$ and $z \in [1, D_z]$. Let us have a look of how the i^{th} activation map pixels $A^l_{x,y,i}$ is obtained from the previous layer $l-1$



(a)

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

114	328	-26	470	158
53	266	-61	-30	344
403	116	-47	295	244
108	-135	256	-128	344
314	346	279	153	421

(b)

Figure 3.7: *(a) Example of a convolution of a* 8×8 *2D image (blue) with a* $3 \times 3 \times 1$ *filter (green) whose bias equal to 0. We explicitly represent the computation of the pixel* $A_{2,2,1}$*. (b) A* 3×3 *filter applied to an image with padding of 1. [39]*

and filter F_i^l.

We suppose that the previous layer is a collection of N_z 2D images of $N_x \times N_y$ pixels. One pixel of those images can be labelled as $P_{x,y,z}^{l-1}$, with $x \in [1, N_x]$, $y \in [1, N_y]$ and $z \in [1, N_z]$. The dimension of the filter F_i^l will be $k \times k \times N_z$, where k is an hyper-parameter of the network: an odd integer different from one (generally 3). This filter is composed of weights $w_{m,n,o}^{l,i}$ with $m \in [1, k]$, $n \in [1, k]$ and $o \in [1, N_z]$ and a single bias b_i^l. Then the pixel $A_{x,y,z}^l$ of the activation map is :

$$A_{x,y,i}^l = \left(\sum_{o=1}^{N_z} \sum_{m=1}^{k} \sum_{n=1}^{k} w_{m,n,o}^{l,i} \cdot P_{x-\frac{k-1}{2}+(m-1)\,,\,y-\frac{k-1}{2}+(n-1)\,,\,o}^{l-1} \right) + b_i^l. \tag{3.4}$$

This equation looks complex, but graphically it is very easy to visualise what F_i^l does to the previous layer. The filter which takes the entire depth of the input volume is convoluted across the width and the height of the input volume, computing the dot product between the entries of the filter and a sub-portion of the input image, centred into the desired destination pixel. Figure 3.7a provides an example for $N_z = D_z = 1$, a $3 \times 3 \times 1$ filter and a bias equal to 0. It represent the computation of the pixel $A_{2,2,1}$.

This way, we will face a problem for the computation of the pixels on the edges of the activation map. We can solve this problem by the addition of a padding to the input image. This consists of a contour of "fake" pixels (with value equal to 0) that surround the input image. The number of needed padding pixels is $P = \frac{k-1}{2}$. Figure 3.7b provides an example of convolution computation with padding.

We have shown here a very simple convolution operation to obtain the activation map, but one can imagine other hyper-parameters to make it more complex. For example the stride S: for the moment, to compute each $A_{x,y,i}$, we have scanned the filters over the input map with a step of one pixel at a time. But filter can be translated by S pixels at a time per output. This leads to smaller $D_x \times D_y$ output dimensions, that can be computed with the following formula:

$$D_\nu = \frac{N_\nu - k + 2P}{S} + 1, \tag{3.5}$$

where $\nu = x, y$. With $k = 3$ (then $P = 1$), and a stride of 1, the output dimensions are the same as the input.

With this method, we have reduced drastically the number of weights and biases of our network. For example, to go from the input layer to the first layer, if the input is a RGB 32×32 image and the first layer is composed of 8 activation maps ($D_z = 8$), we only need 224 weights and biases, instead of the 24584 a classical ANN with 8 neurons in the first layer would need.

Coordinate Convolutional layer

We will not use Conv for the CNN developed for this thesis project, but CoordConv which is a simple extension of the standard convolutional layer. For some specific tasks, it can help a lot algorithms to learn efficiently. The only thing that differs between CoordConv and Conv is the translation invariance. CoordConv is not necessarily translation invariant, so with CoordConv, the network will be able to discriminate two pictures of the same object, if the latter is spatially translated.

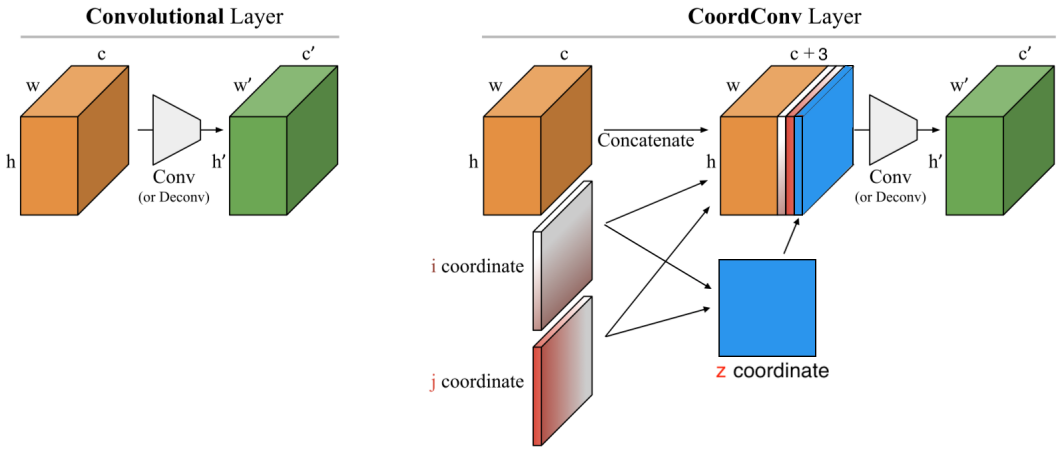


Figure 3.8: *Comparison of Conv and CoordConv layers. (left) A standard convolutional layer maps an input with shape $h \times w \times c$ to as output of shape $h' \times w' \times c'$. (right) A CoordConv layer has the same functional signature, but accomplishes the mapping by first concatenating extra activation maps to the input [40].*

The CoordConv can be implemented as a simple extension of Conv. If the input layer is a set of c activation maps, of dimension $w \times h$, then one needs to add 3 extra activation maps that we will call I, J, and R ($w \times h$ matrices). I will give the row information, J the column information, and R information on the distance between the pixel and the centre of the image. They are computed as follow:

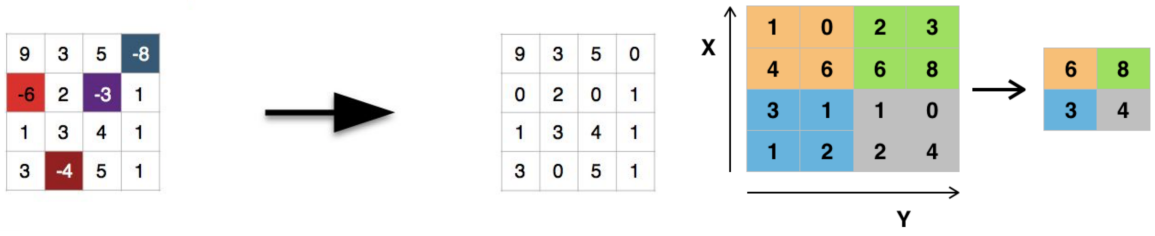
$$\begin{aligned} I_{x,y} &= x \\ J_{x,y} &= y \\ R_{x,y} &= \sqrt{\left(x - \frac{h}{2}\right)^2 + \left(y - \frac{w}{2}\right)^2} \end{aligned} \tag{3.6}$$

with $x \in [1, w]$ and $y \in [1, h]$. Then we apply a linear scaling of the I, J and R coordinate values to make them fall in the $[-1, 1]$ range. Finally, I, J and R are concatenated to the input activation maps and a standard convolutional layer is applied. Figure 3.8 depicts the whole process.

A full description of this layer is given in [40], which also shows that including CoordConv can boost performances of the algorithm in some specific cases.

Rectified linear unit layer

A ReLU layer applies the non-saturating activation function $f(x) = \max(0, x)$ to each pixel of the previous layer (without changing the layer dimensions). Removing negative values increases the nonlinear properties of the cost function. A very simple example of an application of a ReLU layer over a $4 \times 4 \times 1$ input dimension is shown in figure 3.9a.



(a) (b)

Figure 3.9: *(a) Example of an application of a ReLu layer over a* $4\times4\times1$ *input dimension. (b) Example of an application of a max-pooling layer over a* $4 \times 4 \times 1$ *input dimension [39].*

Batch Normalization

The idea behind batch normalisation is to avoid that one pixel of the network becomes increasingly larger than other ones, unbalancing the overall network and inducing bad predictions. A BatchNorm will normalise each pixel of the previous layer by subtracting from this pixel the "batch mean" and dividing it by the "batch standard deviation". The output layer will have the same dimension as the input.

We recall that a "batch" denotes the number $\lfloor \frac{t}{n_{mb}} \rfloor$ of images from the training data set that are used for calculating a gradient step during a training epoch (section 3.1.2). During the Mini-Batch Gradient Descent, weights and biases in Conv layers are adjusted to minimise the cost function, and can be affected by the normalisation of the BatchNorm layer. So we need to add two trainable parameter vectors for each layer: the "standard deviation" parameter vector $\overrightarrow{\gamma}$ and the "mean" parameter vector $\overrightarrow{\beta}$, with $\overrightarrow{\gamma}$ and $\overrightarrow{\beta}$ being vectors of size D_z (depth of the layer). Then, the compensation of this normalisation will be done by changing only those two parameters during the gradient descent step, instead of losing the stability of the network by changing all the weights and biases. The output pixel $A'_{x,y,z}$ after the batch normalization of pixel $A_{x,y,z}$ can be computed as follow:

$$A'_{x,y,z} = \frac{A_{x,y,z} - E[A_{x,y,z}]}{\sqrt{Var[A_{x,y,z}] + \epsilon}} * \gamma_z + \beta_z$$

Where ϵ is a small number (hyper-parameter), $E[A_{x,y,z}]$ and $Var[A_{x,y,z}]$ are the mean and the variance of the $A_{x,y,z}$ pixel computed over the batch. The default value of γ is 1 and default value of β is 0. A deeper explanation is described in this paper by Sergey Ioffe and Christian Szegedy [41].

Pooling layer

The aim of a Pool is to downsize the x and y dimensions of the previous layer, without changing the z dimension. This is done in order to reduce the number of image pixels before entering the fully connected layer part, and so reducing the computational complexity of the algorithm to avoid overfitting. In practice, it partitions the input volume $D_x \times D_y \times D_z$ into a set of non-overlapping volumes, and, for each of those volumes, the

maximum pixel is provided as output (max-pooling). It is common to periodically insert a max-Pool between successive Convs in a CNN architecture.

In most CNNs, those non-overlapping volumes have a dimensionality of $2 \times 2 \times D_z$ (always spread along the Z dimensions of the input). Figure 3.9b gives an example of an application of a max-Pool over a $4 \times 4 \times 1$ input image, so we have 4 non-overlapping areas (in orange, green, blue and grey). One needs to ensure that the dimension D_x (D_y) of the input is an even number, otherwise the addition of a padding layer at the end of the x (y) direction is needed. For $2 \times 2 \times D_z$ non-overlapping areas, the output x and y dimensions of the layer will be divides by 2, so it will scale the D_z activation maps down to 25% of their original size.

Max-pooling is not the only Pool that we can find in CNN architectures, even though it is the most convenient, computationally speaking. But the goal of a Pool remain the same: downsizing the x and y dimensions and keeping the same D_z. One can also find functions that compute the average number of the non-overlapping volumes (average-pooling), or the L1/L2-norm (L1/L2-pooling).

Fully connected layer

Finally, after a combination of Conv, Relu, BatchNorm and max-Pool layers, the x and y dimensions of the output volume should have substantially shrunk compared to the initial input image. Then, all the pixels are given as input of fully connected layers, with an architecture similar to the one we presented in figure 3.1. Those fully connected layers will finally connect to the output layer: the predictions of the algorithm.

3.3 Graphics Processing Units

As we have seen, training an ANN or a CNN is a process which involves a lot of computation, because it requires exposing the model to thousands or millions of data sets. This is highly demanding in computing power and can be time-consuming. Luckily, computer scientists have developed tools to speed up this process.

A Graphics Processing Unit (GPU) is a microprocessing chip originally designed to handle graphics in computing environments, such as calculating and printing 3D content to screen. Contrary to Central Processing Units (CPU) that only have few cores, GPUs can have hundreds and even thousands, but GPU cores are less powerful and run at lower speeds. Thanks to those multiples cores, they can run multiple computations simultaneously as long as the calculation to be performed is parallelizable, making them much faster than CPUs for certain specific tasks.

Computations during the training of ANN can be parallelized, so I thank the LPHE laboratory for providing me a CUDA Nvidia GeForce GTX 1050 Ti, a GPU with 4GB of RAM, which saved me a lot of time. For our project, we used the PyTorch library to store and operate on homogeneous multidimensional arrays of numbers (similar to NumPy Arrays). Pytorch also provide tools to easily compute gradients with PyTorch autograd.

To manipulate data, we use the pandas library. Pandas dataframes are data structures widely used for machine learning, because they allow very easy manipulation of data and analysis of large samples.

Chapter 4

Algorithm for energy reconstruction of an electron crossing a tracker

In this section, we will present a ML algorithm able to recover the energy of an electron which has crossed a tracking detector interlayed with emulsion bricks. We want this detector to be resembling the SND SciFi tracker that will be used for SHiP, so that, in the future, we could use this algorithm on real data.

This algorithm can be very useful for SHiP, as it allows reconstruction of the energy in real time, complementing to the emulsion bricks, which are more accurate but need to be taken out and developed in order to access information. Electrons passing through such a detector will induce electromagnetic showers due to the presence of emulsions bricks acting as passive material. Those narrow "sprays" of particles tracks are extremely challenging to reconstruct with classical tracking algorithms. With deep learning algorithms, once the network is well trained, very fast predictions can be made regarding the features of the particle which crossed the detector. We need a very large data set to train and test the algorithm, so we used FairShip to produce samples of simulated events. The algorithm presented here builds on the basis of S. Shirobokov's algorithm available at [42].

We will first give an overall description of the algorithm, by presenting the two different data set we used (DS1 and DS2), the way data are processed to be fed as input to a CNN, and the inner structure of the CNN. Then we will see a first study, using DS1, to characterize the performance of the algorithm, and finally how this algorithm could be applied to the SND@LHC pilot run.

4.1 Overall description

The task of the algorithm will be to measure the energy of an electron which crosses the TT. The electron will produce an EM shower thanks to the emulsion bricks in between the TT planes, and the hits detected by those planes are used as inputs by the algorithm.

We will use two different data sets as part of this project. The first data set, that we will call DS1, is the original one used for the development of [42]. The second data set (DS2) was produced by me, and aims at reproducing at best the SND@LHC pilot run detector. After building a realistic simulation of SND@LHC pilot run detector in FairShip, we have fired n_{events} (around 500000) of electrons through this detector, and collected the data in a root file.

The tracker digitization is not yet implemented in FairShip at the time of writing, which means that we only have access to the exact position of the TT hits. In addition, the fact of using Monte Carlo simulated data gives us all needed information (the exact position, the exact momentum, the name...) of the particle which made that hit. In reality, we will just know that a charged particle has crossed that plane with a limited spatial resolution. As a CNN uses a set of 2D images as input, we need to transform those raw data into readable input images for a CNN algorithm, which are compatible with the hits information we can get from a SciFi tracker. We explain hereafter how we proceed.

4.1.1 Data preprocessing

During one event, which corresponds to a single electron fired through the tracker, this electron will produce an EM shower in the regions where the emulsion bricks are located. The particles belonging to this shower will cross n_{planes} tracker planes of dimension $d_x \times d_y \times d_z$, producing hits on the different tracker planes. These hits will have (x, y, z) coordinates (raw data in the root file), that are within the range of the TT plane:

$$\begin{aligned} &x \in [-\frac{d_x}{2}, \frac{d_x}{2}] \\ &y \in [-\frac{d_y}{2}, \frac{d_y}{2}] \\ &z \in [z^k_{plane} - \frac{d_z}{2}, z^k_{plane} + \frac{d_z}{2}], \quad \text{for} \quad k = 1, ..., n_{planes} \end{aligned} \tag{4.1}$$

with z^k_{plane} being the z position where is located the k^{th} plane.

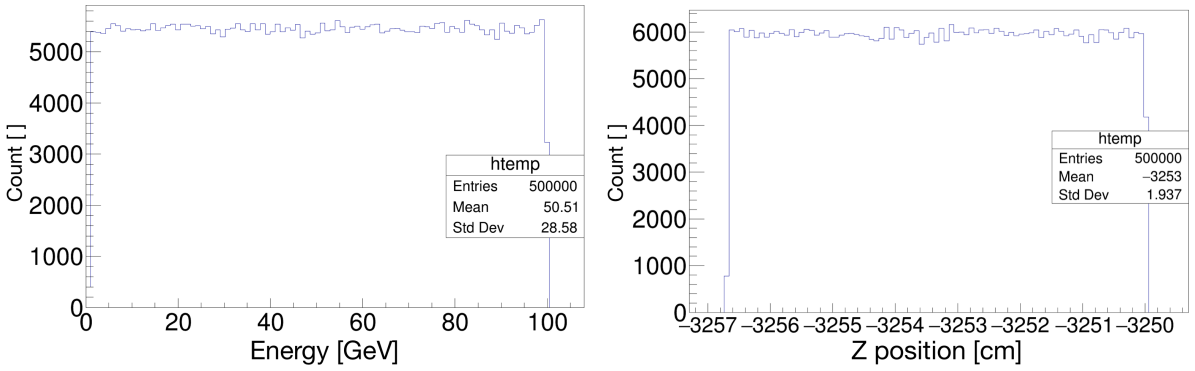
We want these hits to be used by the algorithm as inputs to reconstruct the energy of the electron. As the algorithm is a CNN, the input needs to be a set of 2D images, with a resolution of $n_x \times n_y$ pixels (n_x and n_y are hyper-parameters of the algorithm). So we need to make each (x, y, z) coordinate of a hit match with the $(i, j)^{th}$ pixel of the k^{th} plane. The pixel should be white (equal to 1) if it corresponds to a hit and black (equal to 0) if not. This is the kind of information an actual SciFi tracker can provide, as we will just know that a particle has crossed that plane with a limited spatial resolution.

So technically, for a hit with coordinates (x, y, z), if z belongs to the interval $[z^k_{plane} - \frac{d_z}{2}, z^k_{plane} + \frac{d_z}{2}]$, then the hit corresponds to the k^{th} plane. The hit belongs to the pixel (i, j) of this plane if x and y are respectively in the range $[\frac{d_x}{n_x}(i-1), \frac{d_x}{n_x}i]$ and $[\frac{d_y}{n_y}(i-1), \frac{d_y}{n_y}i]$.

To sum up, for each event, the input will be a set of n_{planes} 2D images, in which each white pixel on the k^{th} image is a hit (or more if multiple particles has crossed the same pixel). We can see on figure A.3 for $n_{planes} = 6$ an example of what an input looks like for different resolutions $n_x \times n_y$.

4.1.2 Data sets

We will present here the two different data sets DS1 and DS2 that we used.



(a) (b)

Figure 4.1: *Histogram of the energy (a) and the position from where electrons were fired (b) for DS1.*

DS1

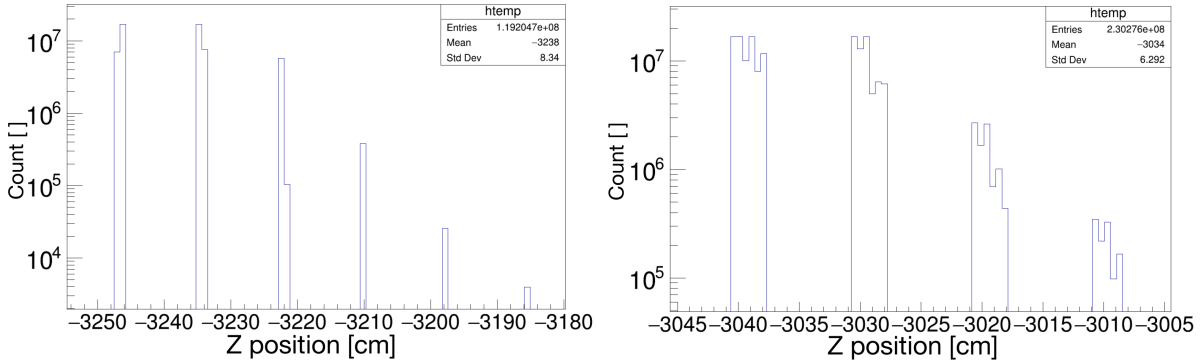
Training a ML algorithm can take a lot of time. This is why, during the development phase, it is convenient to start with a smaller, simplified dataset before switching to a more realistic one. This initial dataset matches exactly these requirements, and it has been used to develop and commission the algorithm as well as to tune its hyperparameters.

The geometry of the detector in this dataset is $n_{planes} = 6 planes$ of dimension $d_x =$ 12.9 cm , $d_y = 10.5$ cm and $d_z = 2.33$ cm. The planes are spaced by 12.2 cm (figure 4.2a) and the emulsions in between (with the same $x - y$ dimension) are 5 cm thick. As an ECC bricks is made of emulsion films and lead plates, we will assume the thickness of the emulsion correspond to 9 times the radiation length X_0 of lead (0.56 cm). The total number of electrons fired is $n_{events} = 500000$. Electrons are fired with a starting position uniformly distributed within a range from 3 to 10 cm before the first TT layer (see figure 4.1b). The initial position of the electrons is varied in order to expose the CNN to showers initiating at various depths. Note that the first emulsion layer is located right upstream of the first SciFi plane, which means that electrons are fired within the first emulsion brick, so that the EM shower can be initiated before the first layer. Fired electrons have an energy uniformly distributed within a range of $[0.3 - 100]$ GeV (figure 4.1a). It is important to uniformly cover the energy range of interest, with the training sample, in order to be able to well reconstruct events with energies that fall in the tails of the data distribution.

Figure A.1a and A.1b in the additional material (chapter A) show, respectively, histograms of the x position and of the y position of the hits detected by the tracker planes. As we expect, both are Gaussians centred in 0 (electrons are fired straight towards the centre of the detector) with a standard deviation of about 1.2 cm, which corresponds approximately to the Moliere radius of lead.

Before being preprocessed as explained is section 4.1.1, the events are selected based on the following criteria:

1. we removed all the hits associated with an uncharged particle, as it is impossible for the SciFi tracker to detect such particles;



(a) (b)

Figure 4.2: *Histogram of the Z position of the hits detected by the SciFi tracker planes for DS1 (a) and DS2 (b). The y axis is in logarithmic scale.*

2. we removed all the hits associated with a particle with energy lower than 100 eV for the same reason;

3. we removed all the hits associated with a particle with a negative momentum, which means that they are scattered backwards;

4. we removed all events with less than 5 hits, which we set as lower threshold in order to attempt the reconstruction of a shower.

Figure A.1c and A.1d show the same histogram as figure A.1a and A.1b, but after this cleaning process. One can see that sigma of the fitted gaussian is reduced by 10% (about 0.1 cm less). Thanks to this process, we removed around 19% of the "raw" hits, so input data became lighter.

DS2

The goal of the project was to develop an algorithm to reconstruct the energy of showering electrons, applying it to an SND-like detector. We choose to base us on the geometry of SND@LHC Pilot run detector, because it only has $n_{planes} = 4$ planes, which are anyway sufficient to contain EM showers in the considered energy range.

Indeed, the graph of the energy loss of an electrons in matter (figure A.4) shows that there are two main effects to take into account: ionization and radiation (Bremsstrahlung). The critical energy E_c is the energy where Bremsstrahlung starts to dominate and $E_c \simeq \frac{580 \text{ MeV}}{Z}$ ($E_c \simeq 7$ MeV for electrons in lead). Beyond this energy, electron will induce an EM shower: the electron will emit a Bremsstrahlung photon after a distance X_0 on average, and the photon will induce pair production ($\gamma \to e^- e^+$) after $\frac{9}{7}X_0$ on average, etc. Which means that after t radiation lengths electrons and photons of the EM shower will have an energy of $E(t) = \frac{E_0}{2^t}$. The shower continues until the critical energy is reached (no more Bremsstrahlung and pair production), after a distance $t_{max}X_0$, where $t_{max} = \frac{\ln(E_0/E_c)}{\ln 2}$. So the length of the EM shower should not exceed 13.8 X_0 (7.84 cm in lead) after the start of the shower. Once the electrons and photons are at low energies

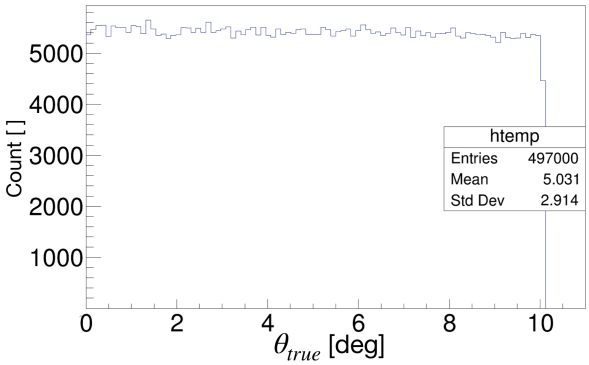


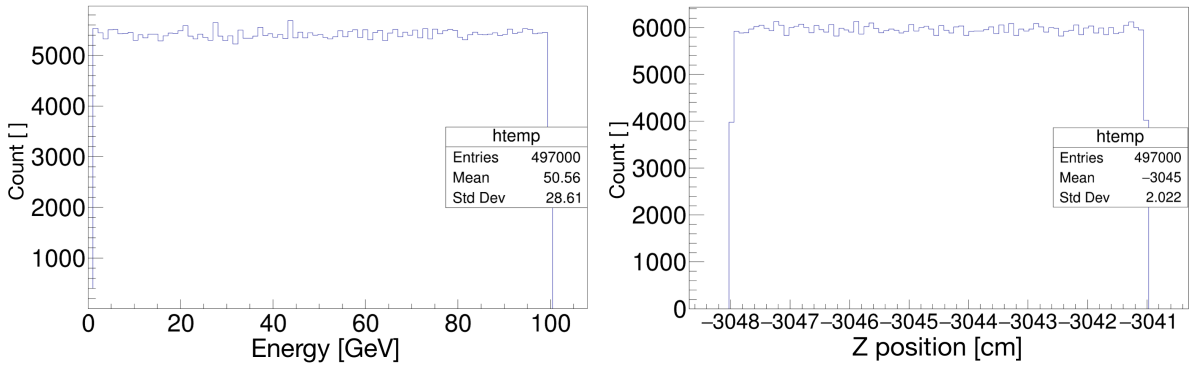
Figure 4.3: *Histogram of the angle of emission of the fired electrons for DS2.*

they continue to loose energy by scattering (Moller, Bhabha) and ionisation, which results in a large number of low energy photons. Figure 4.2a shows that, for DS1, most of the hits are located in the first 4 planes: only 0.02% of the total hits are located in the last 2 planes.

The simulation that we do in order to produce DS2 is very close to the real geometry of SND@LHC pilot run detector, presented in figure 2.8a. An accurate detector description has been implemented, using the materials of which the SciFi planes are made of, and taking into account the geometry of the fibres, arranged in 6 fibres layers per TT plane and sandwiched between honeycomb support structures. The dimensions of the 4 planes are $d_x = 41.6$ cm , $d_y = 38.7$ cm and $d_z = 2.3$ cm. The planes are spaced by 9.35 cm (figure 4.2b) and the emulsions in between (with the same $x - y$ dimension) are 5.6 cm thick, which corresponds to 10 times the radiation length X_0 of the emulsion bricks. The first emulsion layer is located upstream of the first SciFi plane.

The total number of electrons fired is $n_{events} = 497000$. Electrons are fired from an initial position uniformly distributed within a range from 1 to 8 cm before the first TT plane (see figure 4.4b), with a an energy uniformly distributed in the $[0.3 - 100]$ GeV (figure 4.4a). Contrary to DS1, fired electrons of DS2 have an emission angle uniformly distributed within a range from 0° to 10° (see figure 4.3). Indeed, in reality, electrons entering the SND@LHC detector will not necessarily go straight to the centre, but maybe towards the edges of the detector, so we also wanted to check the ability of the algorithm to reconstruct such kind of events. We chose the 10° upper limit by making sure that the cylinder volume of the EM shower are contained within the detector, in order not to loose too much information.

Figure A.2a and A.2b in chapter A show the histogram of the x position and of the y position respectively of the hits detected by the tracker planes, after the same cleaning process that we do for DS1. The distributions are centred in the middle of the TT planes, as the angle is randomly distributed, and we can see that hits are are indeed more spread out compared to DS1: the standard deviation is around 1.70 cm for DS2, compared to 1.1 cm for DS1.



(a) (b)

Figure 4.4: *Histogram of the energy (a) and the position from where electrons where fired (b) for DS2.*

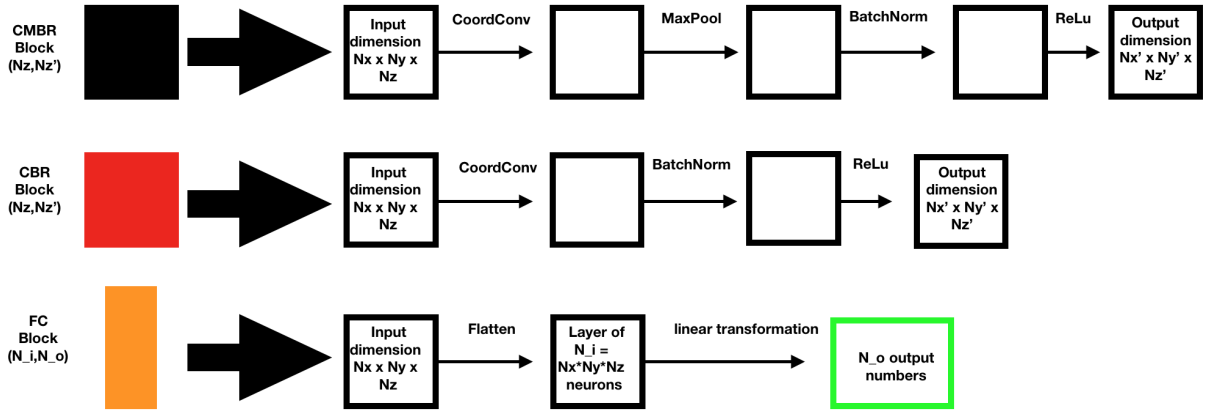


Figure 4.5: *Sketch of the different block structure that we implemented in the CNN.*

4.1.3 CNN architecture

Here, we will briefly explain the overall architecture, and its possible modulations that we have chosen to explore during the optimisation process. We have built three different kind of blocks: two Convolutional blocks (CMBR and CBR), and a Fully connected block (FC), presented on figure 4.5. Those blocks will be used for the construction of the different CNN architectures that we will explore throughout this study, by placing them one after the other to form the network.

A block is simply a combination of CNN layers, and each of the layer that we used has been explained in section 3.2.2. Convolutional blocks blocks must be placed in the upstream part of the network, as they take and return images. They both have two parameters which must be specified: an entry parameter, which is the N_z dimension of the input (i.e. the number of 2D images), and an output parameter, which is the desired N_z dimension of the output.

At the end of the CNN, a final Fully connected block must be placed to compute the

predicted outputs $\overrightarrow{O}_{pred}$ from the last set of 2D images. We also need to specify two parameters for this block. The entry parameter, which is the total number of output pixels $N_i = N_x \times N_y \times N_z$ of the last block, and the output parameter, which is the desired number of predicted outputs N_o (size of $\overrightarrow{O}_{pred}$). For our study, N_o will always be 1 as we only look for energy reconstruction, but one can see that it is very easy to implement other features to recover.

When arranged in a CNN, each block must have an entry parameter compatible with the output of the previous block. This will not be a problem for Convolutional blocks, as we simply need to match the the input z dimension of the block with the output z dimension of the previous one. But when we get to the fully connected part of the CNN, we need to know which x and y dimension is the output of the last convolutional block to know the number of entries of the fully connected block. And to get x and y output of the last convolutional block, we need to know x and y output of the previous one, etc. To make things easier, we created a short python program call "resolution.py", available in the appendix, which, taking the dimension of the input image and the network architecture as inputs, computes N_i, the input dimension of the fully connected block.

To make this program, we use the fact that in CMBR and CBR block, only CoordConv and MaxPool layers change the x and y dimension of the input volume. We use equation 3.5 for CoordConv (with $k = 3$, $P = 0$ and $S = 1$, the hyper-parameter of the CoordConv layer), and for max pooling, we simply take the whole part of the input x and y dimension divided by two.

4.2 Optimisation with DS1

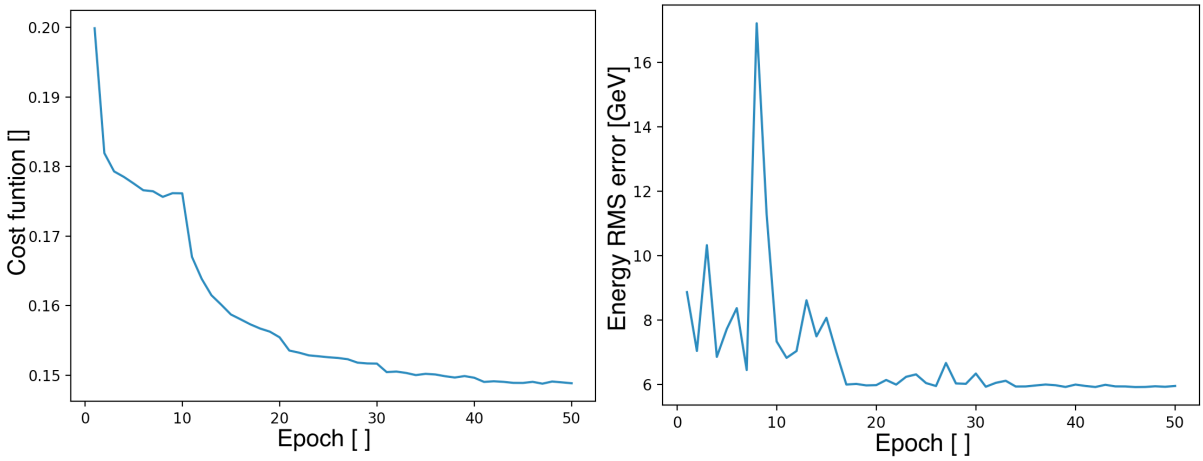
There are plenty of hyper-parameters in a CNN network that you one tune to optimize the performance of the algorithm. We could have chosen the batch size for example, but one can find many studies that show the influence of the batch size on algorithm performances. We choose to take a batch size as large as the available GPU can handle (which correspond to $n_{batch} = 150$). Indeed, a large batch size is better because it can harness the power of GPUs to process more training instances at a time and serve well for scaling and parallelizability as suggested in this article [43].

We could also have chosen to tune the learning rate, but as for batch size, several study have already been performed, especially a learning rate finder technique developed by Leslie N. Smith and presented in his paper titled "No More Pesky Learning Rate Guessing Games" [44]. We choose a standard value for the learning rate equal to 10^{-3}, and it is divided by two every 10 epochs.

The optimisation of this algorithm relies only on two hyper-parameters, which are the structure of the CNN (number of weights and biases) and the resolution of the input image. We choose an easily scalable structure of the CNN, by implementing its layers into blocks. The resolution could be also interesting to tune because it is directly linked to the computing speed of the algorithm.

4.2.1 Example of energy reconstruction

Before doing any optimisation, we start from a basic layout. So we use a very common architecture used in CNN image processing, presented in figure A.6a. It is composed of



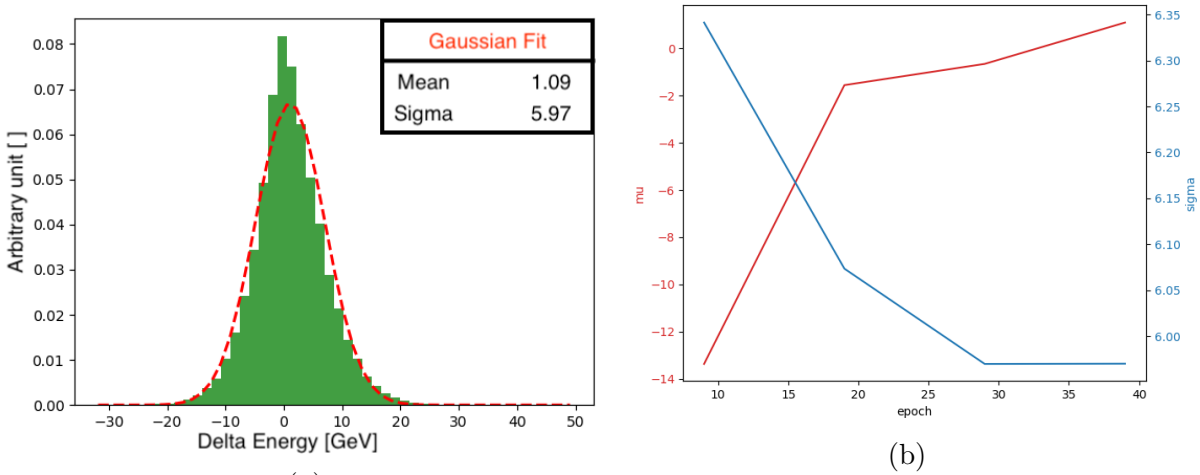
(a) (b)

Figure 4.6: *Cost function (a) and root-mean-square error of the energy on the validation sample (b). The resolution of the input image is* 150×185 *pixels and the CNN has 6 Convolutional blocks.*

6 Convolutional blocks and one fully connected block. We choose for the input image a resolution of 150×185 pixels, which means that a pixel covers a TT area of 700 μm^2. This is the best resolution our GPU can support with this architecture. Note that the spacial hit resolution of a SciFi layer is 250 μm. Figure 4.6a shows the evolution of the cost function in terms of the training epoch. As we expect, the cost function is decreasing, which means that the algorithm is recovering with more and more accuracy the energy of the electrons of the training sample. Figure 4.6b shows the root-mean-square error (RMSE) of the energy, calculated on the validation sample, as a function of the training epoch. This value corresponds to the ability of the algorithm to reconstruct the correct electron energy on unseen data: it is decreasing and oscillating until epoch 20, and it starts to be very stable (close to 6 GeV) from epoch 35. One can also notice a correlation between the bump visible on figure 4.6b at epoch 10 and the fast descent after epoch 10 on figure 4.6a. We can suppose that, at epoch 10, the network has learned the ability to distinguish a new feature of the input images, that helps for the accuracy of the energy reconstruction.

On figure 4.7a, we plot the histogram of the difference between the true energy of the fired electron and the energy predicted by the CNN at epoch 40. One can see that we have a resolution of 5.97 GeV, and the mean is shifted by 1.09 GeV, which means that the reconstructed energy tends to be underestimated. We see in figure 4.8 that this shift is partly due to the CNN's bad ability at reconstructing high energy events.

We have stored the values of the weights and biases of the network every 10 epochs in order to study the evolution of the algorithm accuracy throughout the training process. If one looks at the evolution of the parameters σ and μ of the fitted Gaussian of figure 4.7a over the training epochs (figure 4.7b), it is evident that μ and σ are both getting closer to 0, which confirms the ability of the network to reconstruct the electron energy with increasing accuracy.



(a) (b)

Figure 4.7: *(a) Histogram of the difference between the energy of the fired electron predicted by the CNN at epoch 40 and its true energy. (b) Evolution of the parameters σ and μ of the fitted Gaussian as a function of the training epoch.*

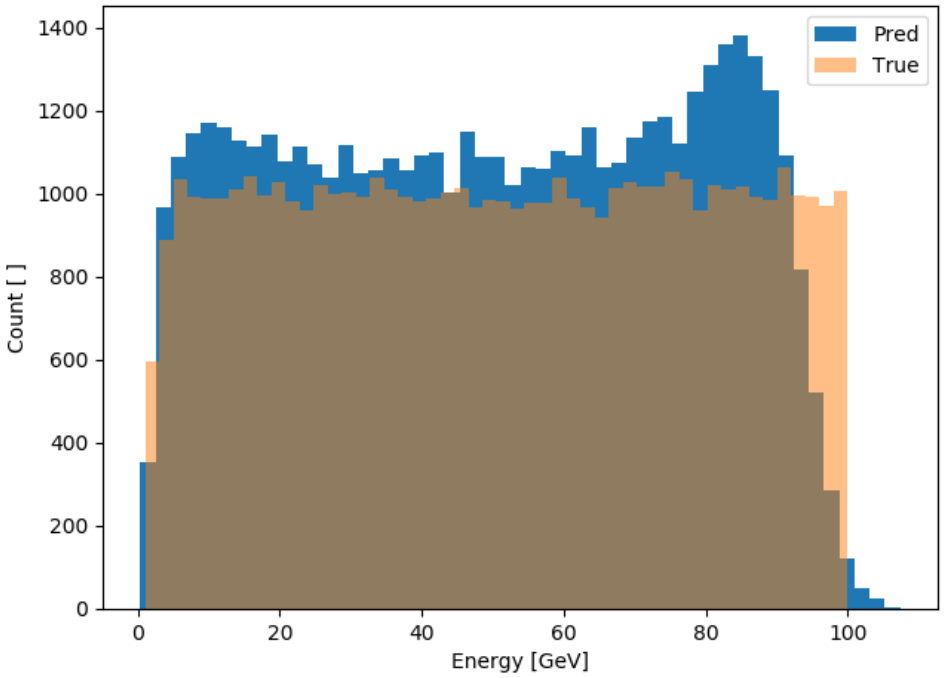


Figure 4.8: *Comparison between the true energy distribution of the simulated electrons and the electron energies predicted by the trained CNN (40 training epochs).*

4.2.2 Tuning hyper-parameters

Now that we have studied the ability of the algorithm to reconstruct the energy, we can start the optimisation phase. We will see the influence of the structure of the CNN and resolution of the input image. With the characteristics of the previous CNN (input resolution of 150×185 pixels and 6 Convolutional blocks), the computing time was very slow. An epoch lasted on average one hour and twenty minutes, which means that we had to wait two days before the CNN was well trained.

So the first things we did were to downsize the resolution of the input images to 100×123 pixels, and to check what was the influence of the number of Convolutional blocks on the algorithm performances. Figure 4.9a shows the RMSE of the energy of the validation set for different CNN architectures. It appears that removing blocks makes the algorithm learn faster: with only three CMBR blocks, the RMSE is stabilised after epoch 20, compared to epoch 26 for an algorithm with 6 CMBR blocks.

Reducing the number of blocks does not necessarily make the training faster, as the weights and biases encoded within the removed Convolutional blocks are compensated by the weights and biases of the fully connected layer. Indeed, the more blocks we remove, the higher the x and y dimensions of the last convolutional block will be (N_x and N_y are reduced by ≈ 2 by each CMBR block).

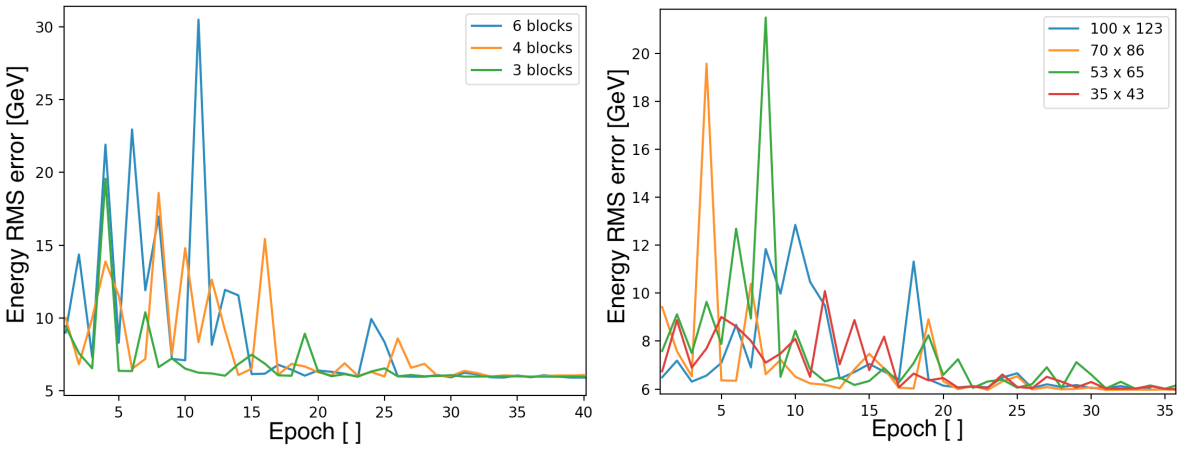
The only modification that helps to make the algorithm faster is to downsize the resolution of the input images. But we need to ensure that downsizing the resolution will not affect the accuracy of the algorithm. Figure 4.9b shows the RMSE of the energy for different input resolutions, all with the same CNN architecture (3 blocks architecture pictured by figure A.6b). We can see on figure A.3 what input images with such resolutions would look like for the same event. It appears that changing the resolution does not affect the accuracy of the algorithm. However, it saves a lot of time: an epoch lasts about 11 minutes for a 70×86 input resolution, instead of 80 minutes for a 150×185 input resolution. It is coherent, as if one reduced the x and y input dimensions by 2, then the number of input parameters is reduced by 4, and so is the computing time.

4.3 Application to the SND@LHC pilot run: DS2

Now that we have studied the performance of the algorithm on DS1, we can start applying the same process to DS2, which corresponds to data simulated with a complete detector description with the layout of the SND@LHC pilot run detector.

4.3.1 Downsizing process

The transverse dimensions of the SciFi planes used to produce DS2 are around 4 times larger than those of the detector used for DS1, which means that the surface covered by those planes is 16 times larger. If we want the resolution per unit area to be the same for both data sets, the input images of DS2 will have 16 times more input pixels than DS1. Unfortunately, our GPU is not powerful enough to handle such input images, which leaves us with two choices: either we drastically reduce the resolution per unit area of DS2, or we keep the same resolution but we extract only a portion of the image, where most of the hits are located. Reducing drastically the resolution seems to be a bad idea,



(a) (b)

Figure 4.9: *RMSE of the energy of the validation set for different CNN architecture (number of Convolutional blocks) (a) and input resolutions (b) as a function of the training epoch.*

as most of the hits in EM shower are located very close to each other, so we would lose much information (regardless if there are 1 or 5 hits on the same image pixel, the pixel value will be 1 anyways). So we chose the second option, and we will explain hereinafter step by step how we proceed.

Figure 4.10 is an explanatory sketch of the downsizing procedure for one event. The first thing to do is to compute the barycentre of all the hits in each plane. It appears that, for each event, most of the hits are located on two planes only: we already saw that the EM shower is theoretically absorbed after 13.8 X_0 on average, and emulsion bricks are 10 X_0 thick in DS2. So, instead of measuring the barycentre of each plane, we only compute the barycentre of the two planes on which there are the most hits (red stars on figure 4.10), and we extrapolate the barycentres of the other two planes (pink stars) by drawing a straight line passing from the two computed barycentres. This line (dotted line on figure 4.10) should be close to the line representing the initial direction of the fired electron (black arrow), as this direction is strongly correlated with the direction of the EM shower. Then, we take all the hits that are located within a sub-image of reduced dimensions (green square), centred on the barycentre of the plane, and we use this reduced image as the input of the algorithm. During this process, a certain number of hits are lost (hits surrounded by red circle).

Figure 4.11 is an example of the image before (images 4.11a) and after this downsizing process (images 4.11b). We also represented the extracted sub-image, the barycentres (computed or interpolated) and the hits that we loose in the process. We see that, in the specific event depicted here, we have reduced by 133496 (i.e. $4 \times (205 \times 196 - 83 \times 82)$) the number of input parameters, by losing only 17 hits.

Until now, we did not discuss the way we compute the dimensions $d_x \times d_y$ of the green square, which should be the same for all events in order to have the same image dimension throughout the inputs. We want this dimension to be sufficiently small to have a small number of input parameters for our algorithm, and sufficiently large to loose only a small proportion of the hits for each events.

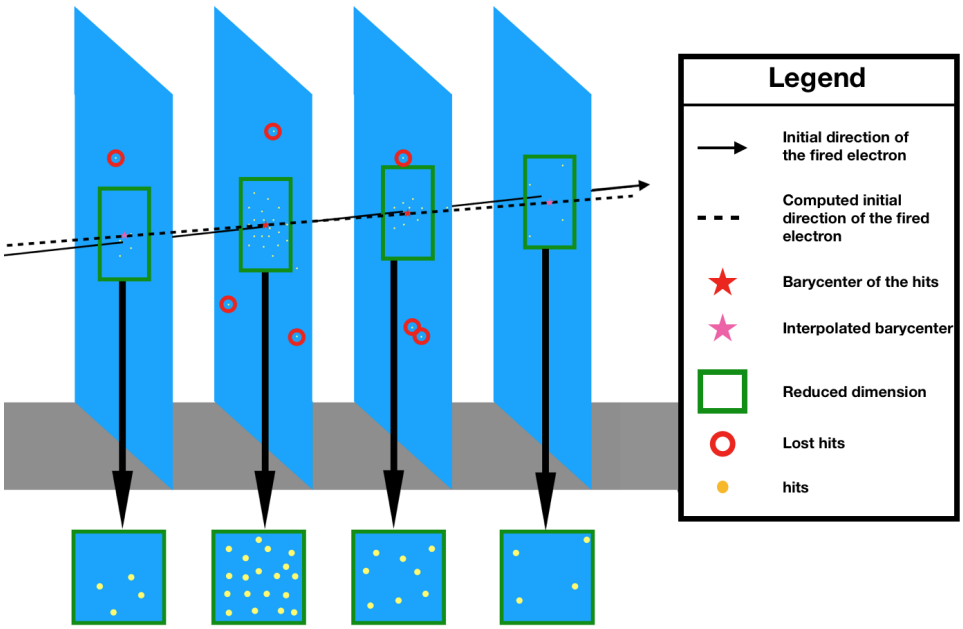
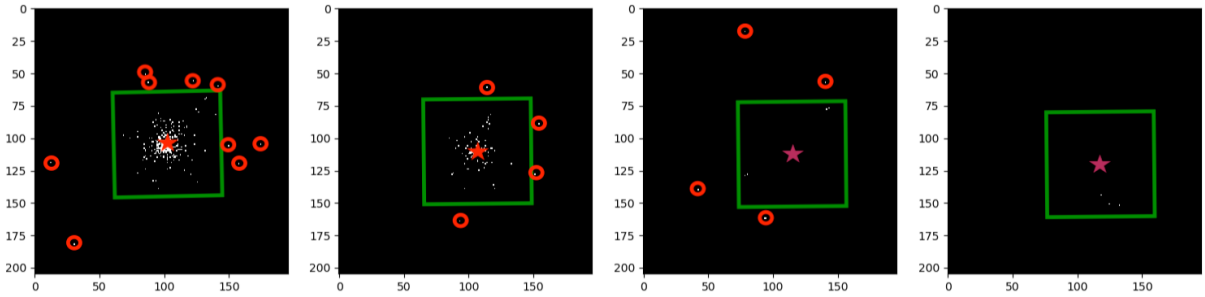
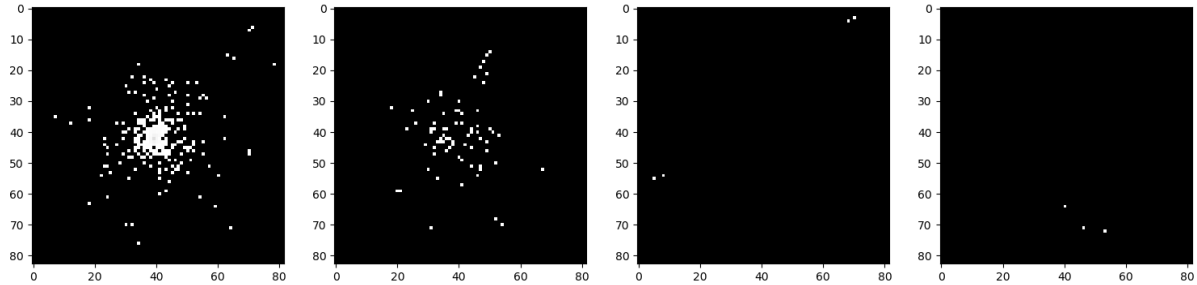


Figure 4.10: *Sketch explaining the downsizing procedure for one event.*



(a) The resolution of the images is 205 × 196 pixels



(b) The resolution of the images is 83 × 82 pixels

Figure 4.11: *Example of the images of the 4 SciFi planes before (a) and after (b) the downsizing process. The legend is the same as that of figure 4.10.*

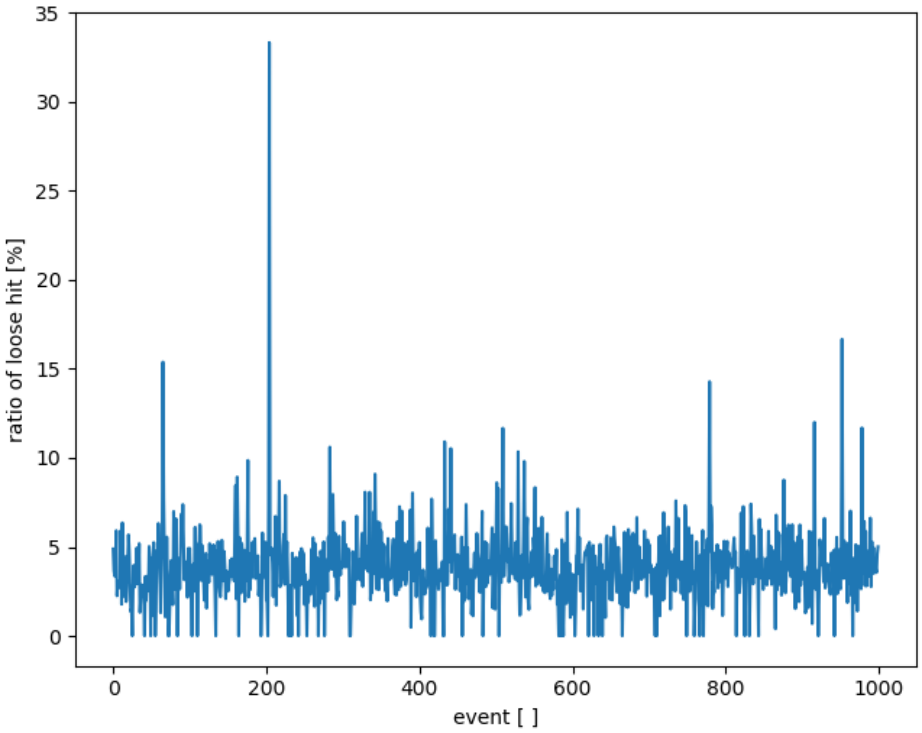


Figure 4.12: *Ratio of hits lost in the downsizing process over a sample of 1000 simulated events.*

Let $B_k^i(x)$ and $B_k^i(y)$ be the x and y coordinates of the barycentre of the k^{th} plane for the event i (computed with the hits, if plane i is among the two planes with the most hits, or with the interpolation technique otherwise), and $h_{k,j}^i(x)$ and $h_{k,j}^i(y)$ the x and y coordinates of the j^{th} hit on the k^{th} plane for the event i. To compute d_x and d_y, the idea is to plot the histogram of the quantity $X_{shifted} = h_{k,j}^i(x) - B_k^i(x)$ ($Y_{shifted} = h_{k,j}^i(y) - B_k^i(y)$) for all k, j and i, and look at the standard deviation of this quantities, which correspond to the x, y coordinates of the hit with respect to the position of the barycentre on the corresponding plane. If the histogram of $X_{shifted}$ is a perfect Gaussian with a standard deviation σ_x and the histogram of $Y_{shifted}$ has σ_y, then 99% of the hits will be located within an area $d_x = 6\sigma_x$ and $d_y = 6\sigma_y$ around the barycentre position. Figures A.5a and A.5b show, respectively, the histogram of $X_{shifted}$ and $Y_{shifted}$.

So if we take the standard deviation of $X_{shifted}$ and $Y_{shifted}$, we have $\sigma_x = 2.622$ cm and $\sigma_y = 2.546$ cm, so theoretically if we want only 1% of lost hits, we need to take $d_x = 15.7$ cm and $d_y = 15.3$ cm. As $X_{shifted}$ and $Y_{shifted}$ data are not perfectly Gaussian, it is wise to take an extra margin of at least 1 cm. In the end, we chose $d_x = 17$ cm and $d_y = 16.5$ cm. Figure 4.12 shows that, for such dimensions, the ratio of hits lost per events is not 1% as we expect but 3.84% (mean over all events). We see on this figure that for some events, this ratio can be close to 35%. Such event correspond either to very spread EM showers, or to events with a small number of hits.

The downsizing process allows us to recover another feature of the fired electron: the angle of emission. Indeed, we can compute the angle of the dotted line on figure 4.10, and compare it with the true angle of emission of the electron. Figure 4.13 is an histogram

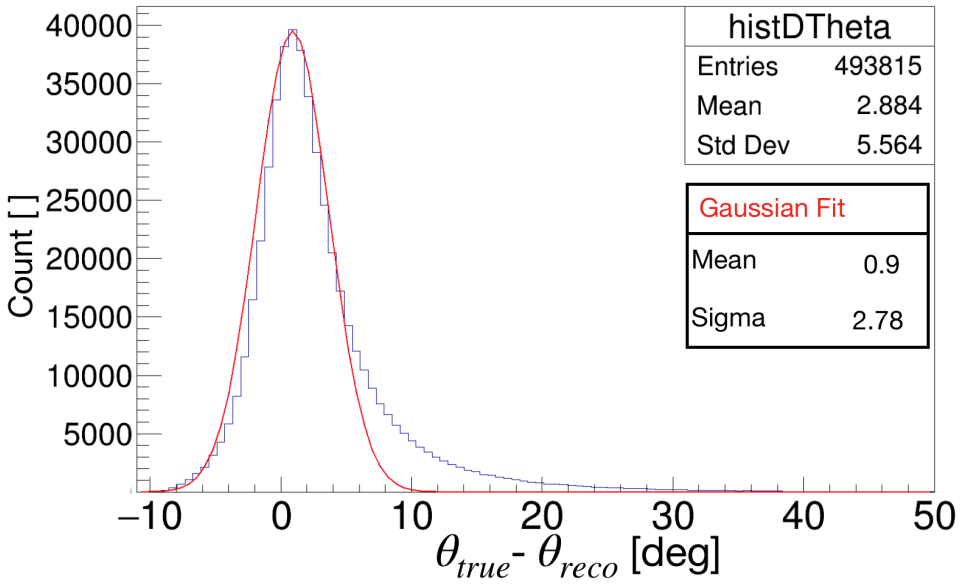


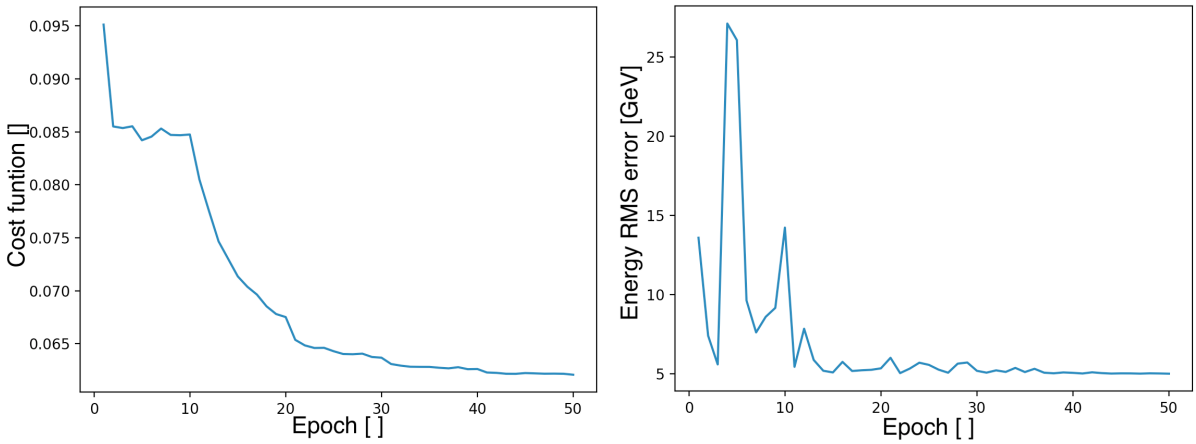
Figure 4.13: *Histogram of the difference between the angle computed using the barycentres of the planes and the true angle of emission of the electron.*

of the difference between the computed angle and the true angle. It looks like a gaussian with a larger tail on the right-hand side. Indeed, even if the maximum angle of emission of the electron is 10°, the shower can still be initiated by secondary particles with an angle larger than 10°.

4.3.2 Performance of the algorithm on DS2

In this section, we present the results we obtained for DS2, in which we apply the downsizing procedure we have described in the previous section. We chose a resolution of the input image of 205×196. After the downsizing procedure, the size of the input images is 84×83, which means that we cut away 33208 pixels. The CNN architecture is presented in figure A.6c. Figure 4.14a shows the evolution of the cost function as a function of the training epoch. As for DS1, the cost function is decreasing, which means that the algorithm is reconstructing the electron energy with increasing accuracy. Figure 4.14b shows the root-mean-square error (RMSE) of the energy on the validation sample as a function of the training epoch. As for DS1, it is decreasing and oscillating until becoming stable (close to 5 GeV) from epoch 30.

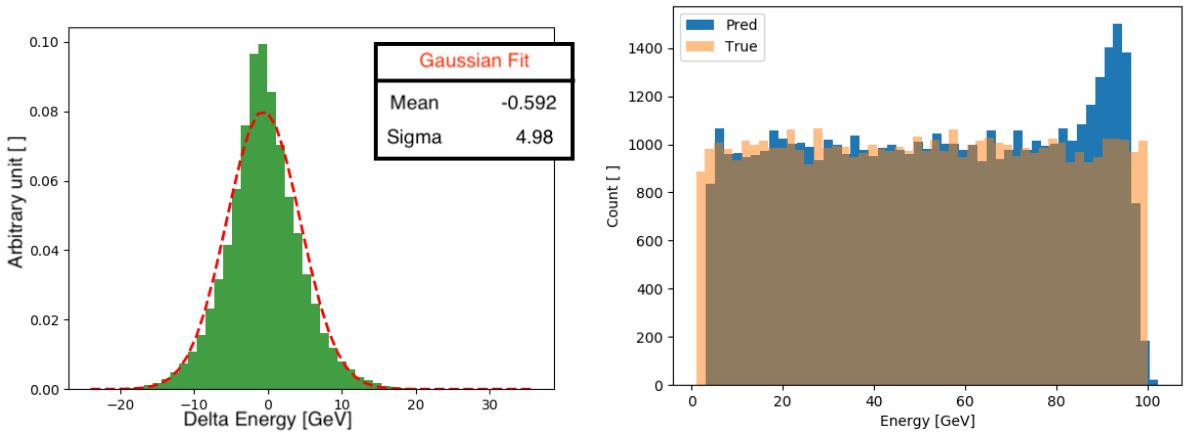
To get a better idea of the accuracy of the algorithm with DS2, we plot on figure 4.15a the histogram of the difference between the true energy of the fired electron and the energy predicted by the CNN at epoch 40. One can observe a resolution of 4.98 GeV, and that the mean is shifted by 0.59 GeV, which means that the reconstructed energy tends to be overestimated. We gain 1 GeV of resolution with DS2, as compared to DS1, which can be explained with the larger size of the detector (more hits taken into account means more information on the electron).



(a) (b)

Figure 4.14: *Cost function (a) and root-mean-square error of the energy on the validation sample (b) for DS2. The resolution of the input downsize image is* 83×82 *pixels and the CNN has 6 Convolutional blocks.*

If we compare the distribution in energy of the electrons, and the energy predicted by the CNN after training with DS1 (figure 4.8) and with DS2 (figure 4.15b), we see that for both data set, high energy electrons (90 to 100 GeV) are badly reconstructed. But contrary to DS1, low energy electrons (0 to 3 GeV) are also less well reconstruct for DS2. Finally, the computation time per epoch is of the same order (15 minutes per epoch).



(a) (b)

Figure 4.15: *(a) Histogram of the difference between the energy of the fired electron predicted by the CNN at epoch 40 and its true energy. (b) Comparison between the distribution in energy of the electron, and the energy predicted by the CNN after training (40 training epochs).*

4.4 Performance of the SND as a sampling calorimeter

4.4.1 Sampling calorimeters

A calorimeter is an apparatus that measures the energy of particles. When a particle enters the calorimeter, it will initiate a particle shower that will be collected and measured. The energy may be measured in its entirety, requiring the whole calorimeter to be made of sensitive material, or it may be sampled at various shower depths. In any case, the calorimeter should be thick enough, in terms of radiation lengths X_0, to contain the whole development of the shower.

We call the calorimeters belonging to the last group "sampling calorimeters" (SC). As only a fraction of the particle energy is measured, the energy resolution of these calorimeters is greatly affected by sampling fluctuations [45]. Usually, the design of a SC is an array of thin counters separated by layers of absorbers. Counters are able to count a certain fraction of the number of charged particles that cross the detector, so that the amplitude of the counter signal is proportional to the total number of charged particles. We can see that the SND can be seen as a SC, with the emulsion bricks playing the role of absorbers, and the SciFi layers playing that of the counters.

One can find in literature [45] a simplified expression for the energy resolution σ_E of a calorimeter:

$$\frac{\sigma_E}{E} = \sqrt{\left(\frac{a}{\sqrt{E}}\right)^2 + \left(\frac{b}{E}\right)^2 + c^2}, \tag{4.2}$$

where a, b and c are detector-specific constants that can be experimentally measured. The physical meaning of the a term comes from the photoelectron statistics of the counters: it models these stochastic fluctuations and it is therefore called stochastic term. The b term accounts for the electronics noise (which can be neglected in our simulation) and the c term is due to calibration uncertainties.

For a SC, where the dominant contribution are sampling fluctuations, the a factor in equation 4.2, one can use the following approximate equation as suggested in [46]:

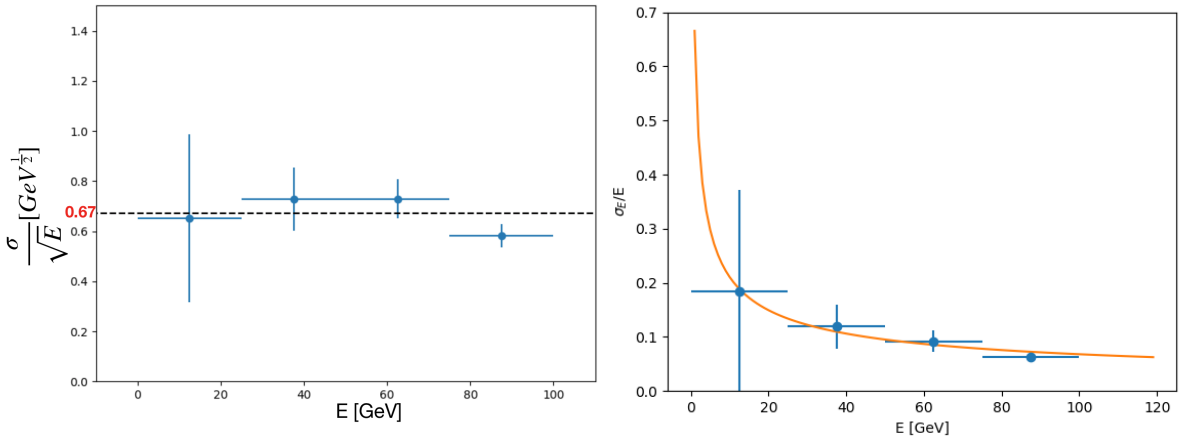
$$\frac{\sigma_E}{E} = \frac{a}{\sqrt{E[\text{GeV}]}}\sqrt{\frac{s[\text{mm}]}{f_{samp}}} \tag{4.3}$$

where a has a typical value of a few percents, s is the thickness of the sensitive layer and f_{samp} the sampling fraction, which is the ratio of ionisation energy loss of minimum-ionising particles in the sensitive layer with respect to the total energy loss in the sensitive layer and absorber. This empirical formula can be used for a preliminary estimate of a and verification of the sampling-calorimeter characteristics.

4.4.2 Real-time energy resolution of the SND

The algorithm we have developed allows us to verify that the simulated SND tracker behaves as a SC.

To compute σ_E, we divide into 4 bins the energy range of the true energy of the fired electrons: $B_1 = [0, 25]$ GeV, $B_2 = [25, 50]$ GeV, $B_3 = [50, 75]$ GeV and $B_4 = [75, 100]$ GeV. Then we took all events which have a true energy within a B_i, and we plot



(a) (b)

Figure 4.16: *Plot of the quantities* $\sigma_E/\sqrt{E}$ *(a) and* σ_E/E *(b) computed with DS2 after 50 training epochs. The orange line in figure (b) is the fit of* $\sqrt{\left(\frac{a}{\sqrt{E}}\right)^2 + c^2}$ *to the data points. The fitted parameters are* $a = 0.665 \pm 0.002$ $GeV^{1/2}$ *and* $c = 0.014 \pm 0.004$.

in an histogram the difference between the true energy and the energy reconstructed by the algorithm after training. We perform a Gaussian fit of these 4 histograms, as we did in figure 4.15a to find σ_{B_i} (the standard deviation of the fitted Gaussian of the histogram corresponding to B_i), and $\Delta\sigma_{B_i}$ the estimated error of σ_{B_i}.

Figure 4.16b shows the relative resolution of the SND, with the SciFi used as sensitive layers through the ML algorithm object of this work, as a function of E. In figure 4.16a one can see the relative resolution multiplied by $\sqrt{E}$. The constant behaviour of these data points, predicted by equation 4.3, confirm that the stochastic fluctuations term is the dominant one for this sampling calorimeter, with a value of approximately $a = 0.67$ if one neglects the other terms of equation 4.2. The orange line in figure 4.16b shows a fit of equation 4.2 to the measured data points, with $b = 0$ as we do not have electronic noise in our Montecarlo simulation. We find $a = 0.665 \pm 0.002$ $\text{GeV}^{1/2}$, which is in accordance with figure 4.16a, and $c = 0.014 \pm 0.004$.

Up to the accuracy available during this study, we therefore conclude that the simulated SND behaves as expected as a sampling calorimeter, with the advantage of being able to provide energy information in real-time.

Chapter 5

Summary and conclusions

This thesis work prooves that it is possible to reconstruct electromagnetic showers using a tracking detector. To this aim, using classical tracking algorithms would be extremely challenging, due to the high hit occupancy of EM showers. We therefore designed and commissioned a dedicated technique using a Machine Learning algorithm with a Convolutional Neural Network. We successfully employed this algorithm to reconstruct the energy of electrons showering through a calorimeter composed of emulsion bricks interleaved with scintillating fibre tracker planes. Only data from the tracker has been used, with the emulsion bricks acting as passive material. We also developed a procedure which allows the algorithm, without decreasing its accuracy, to deal with larger dimension input images, i.e. tracking stations of large surface, such the ones that will be used in the Scattering and Neutrino Detector (SND) of the SHiP experiment. Moreover, the way in which the algorithm is implemented easily allows to reconstruct, at the same time as the electron energy, also additional features of the shower, such as the position of the first vertex, the direction, etc. This possibility can be implemented and studied as a future work.

The relative energy resolution has been measured as a function of the incident electron energy, and its behaviour confirms that the SND, used as real-time calorimeter thanks to the developed algorithm, behaves as a sampling calorimeter with a stochastic term $a = 0.665 \pm 0.002$ GeV$^{1/2}$.

The present study raises some unsolved questions: why the energy is on average slightly overestimated? Why is the energy reconstructed with less accuracy for showers with particularly low (below 3 GeV) or high energy (beyond 90 GeV)? We leave those questions open for future studies, but we can give a possible line of approach. Higher energy showers will have more hits, which means a higher probability that hits accumulate in the same pixel of the image. One can study this hypothesis by increasing the image resolution, which requires a more powerful GPU. An larger training sample with extended energy range should also be used to retrain the algorithm and look for "border" effects.

Moreover, in a realistic scenario, the SciFi Target Tracker (TT) planes will not provide the x and y coordinates of each hit. Each SciFi plane can only measure one coordinate, so we will in principle not be able to tell which x coordinate correspond to which y. When we reconstruct the hit images, we will end up with a certain amount of "ghost hits", which do not match any real particle. This aspect is not implemented in FairShip, and we did not take it into account in this project. However, we can suggest some ways

to make this algorithm deal with ghost hits. A first possibility would be to applying a ghost-removal algorithm, such as the one that is being developed right now in the EPFL LPHE laboratory. Another possibility would be to change the input to the two vectors of x and y hit coordinates rather than feeding the algorithm with 2D images, retrain it and evaluate the algorithm's performance in this case. It will be very interesting to study these different possibilities in the future.

We would like to stress that this work enables the reconstruction of EM showers on the fly using a tracking detector. The application of this work in the SHiP experiment, as well as in SND@LHC, will make it possible to spot early hints of possible presence of dark matter, which would be inaccessible if relying only on the emulsions bricks for calorimetry. Indeed, Dark Matter scattering on the material of the emulsion bricks as a signature similar to that of elastic ν_e scattering. If one is able to identify these kinds of event from other processes with a real-time solution, one is able to count the frequency of ν_e elastic scattering events. The Standard Model allows to calculate the expected frequency of such events in such detector, which means that, by comparing the observed and the expected rates, we are able to tell if there is an excess. Such excess would be a possible hint of dark matter. All of this could be done in real time.

One could go even further, and also think of directly discriminating between ν_e elastic scattering and dark matter scattering. In fact, massive dark matter would induce a different kinematics with respect to massless neutrinos. Such discrimination would require that the energy and the angle of each shower are known with great accuracy.

After a solution has been found for the problem of ghost hits, and the accuracy of the algorithm has been better calibrated at high energies, it can immediately be commissioned using real data such as that collected by the SHiP collaboration at the DESY test beam. This commissioning would make the algorithm ready for implementation in the SND@LHC experiment.

Aknowledgements

I would like to express my deep gratitude to the director of the LPHE-LS Prof. Dr. Lesya Shchutska, who gave me the opportunity to do my Master Thesis on such an interesting project. Her patient guidance and helpful suggestions were precious to me.

I would like to offer my special thanks to Dr. Elena Graverini, for all her encouragements, availability and useful critiques during this research work. Her valuable support allowed me to carry out this project to its end.

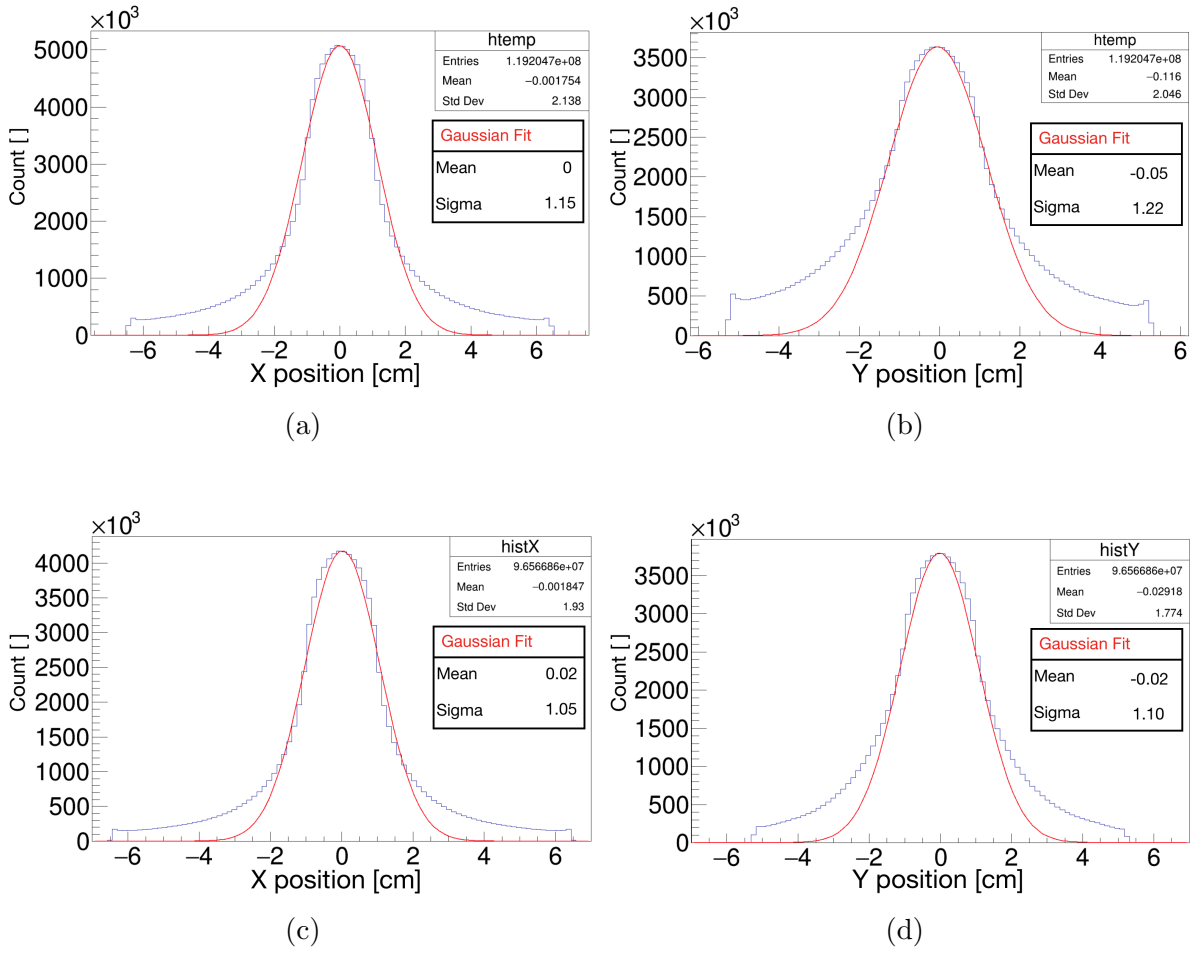
As this project is my first approach to Machine Learning, I asked for help to Sergey Shirobokov, who is a PhD researcher at CERN working in the development of Machine Learning techniques for the SHiP experiment optimisation. I thank him for his very useful help and the share of his knowledge.

I would like to thank in general terms the SHiP team members for welcoming me on board, and all the LPHE members for being amazing and kind co-workers.

Finally, I would also like to thank my family who have always pushed me, unconditionally, in everything I have undertaken, my cat for the fluffy support, all my wonderful friends from *La Meute* along with Elisa, with whom I shared those 5 years of amazing EPFL moments.

Appendix A

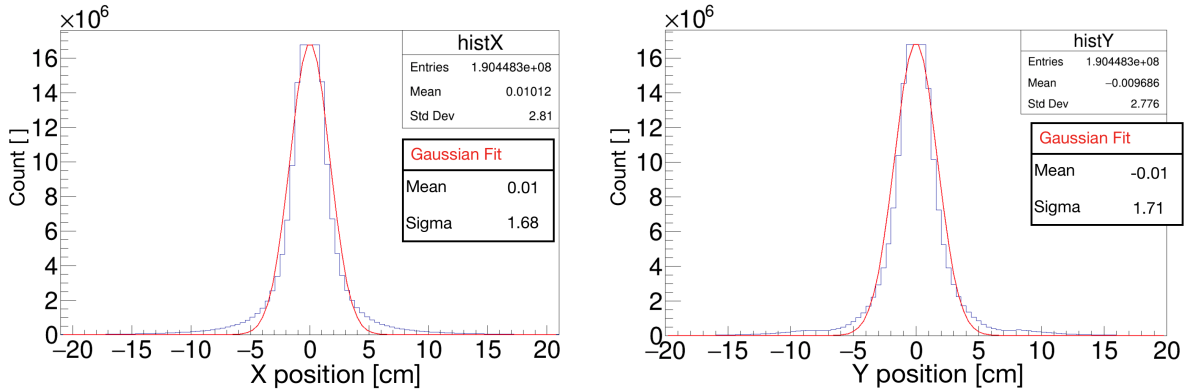
Additional material



(a) (b)

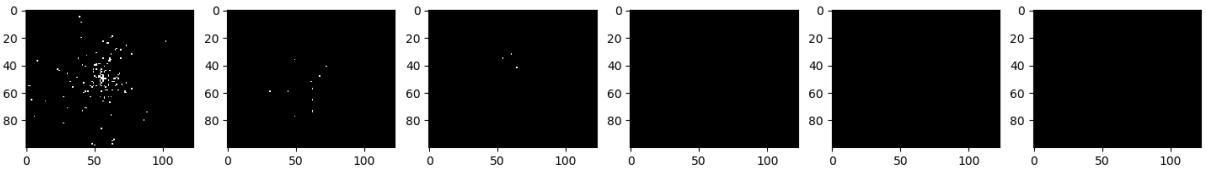
(c) (d)

Figure A.1: *Histogram of the x position (a) and of the y position (b) of the hits detected by the SciFi tracker planes for DS1. The same histograms after the cleaning process described at the end of section 4.1.2 are shown, respectively, in (c) and (d).*

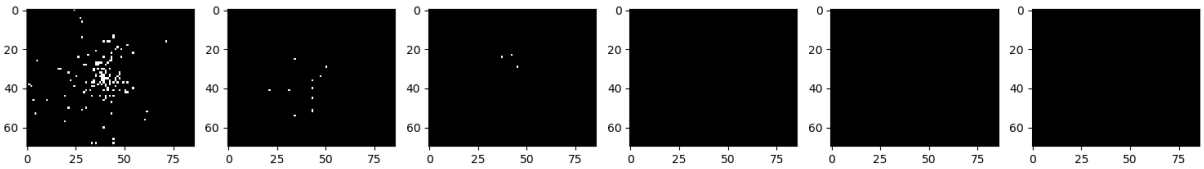


(a) (b)

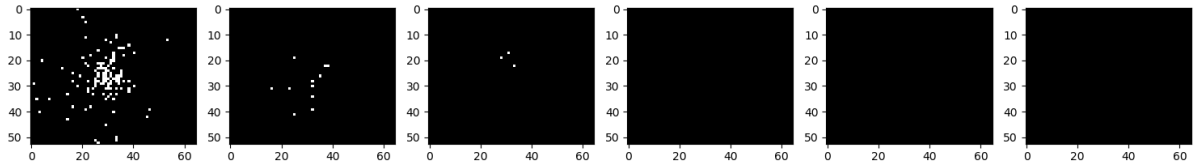
Figure A.2: *Histogram of the x position (a) and of the y position (b) of the hits detected by the SciFi tracker planes for DS2, after the cleaning process described at the end of section 4.1.2.*



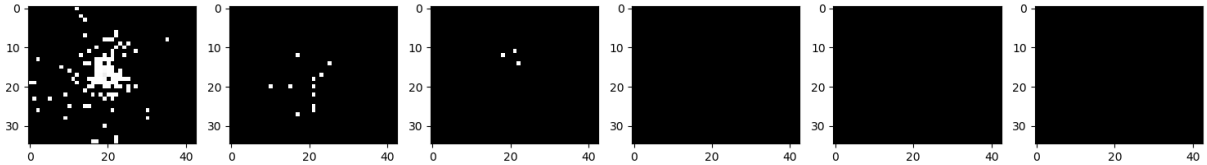
(a) *Resolution* 100×123 *pixels*



(b) *Resolution* 70×86 *pixels*



(c) *Resolution* 53×65 *pixels*



(d) *Resolution* 35×43 *pixels*

Figure A.3: *Example of the same input event of DS1 for different resolutions.*

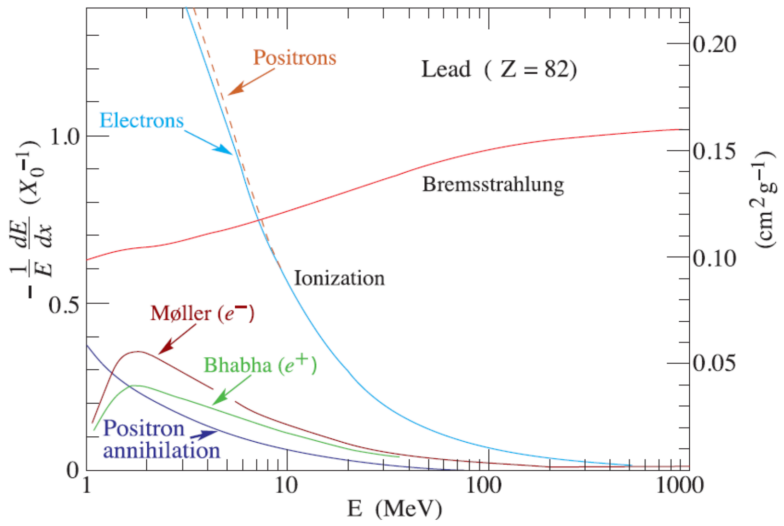
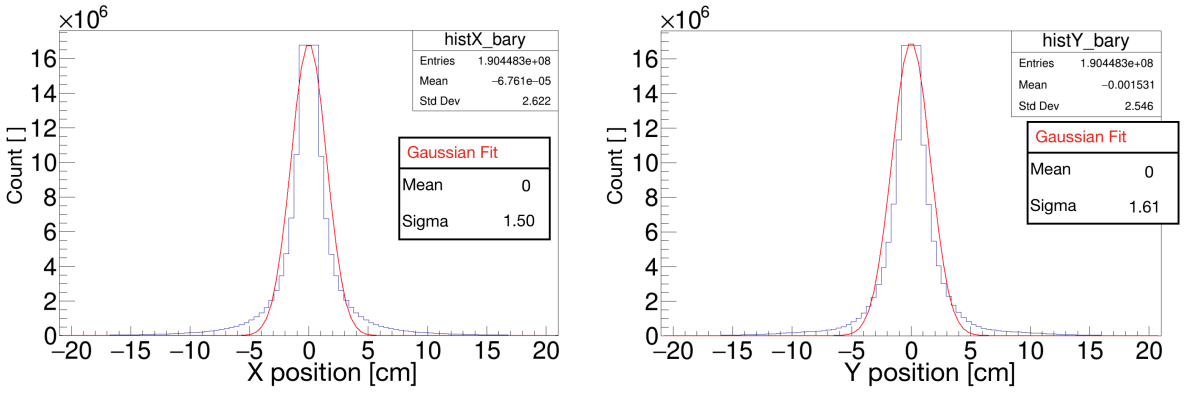
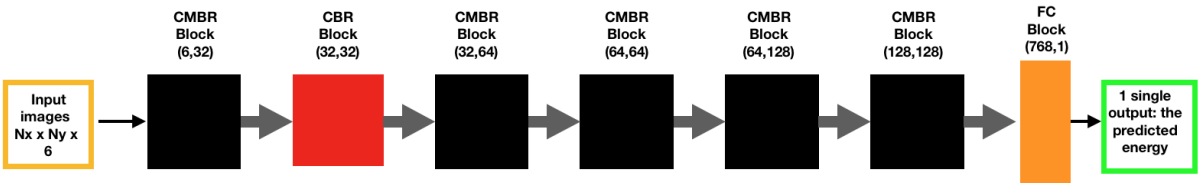


Figure A.4: *Energy loss of an electron in lead ($Z = 82$) per radiation length.*

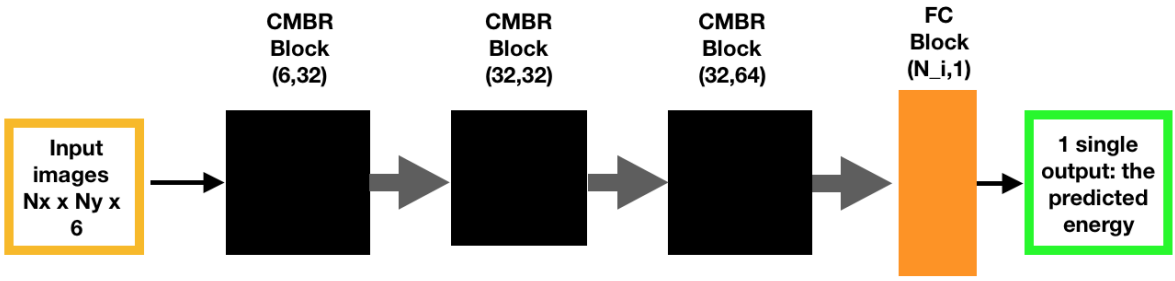


(a) (b)

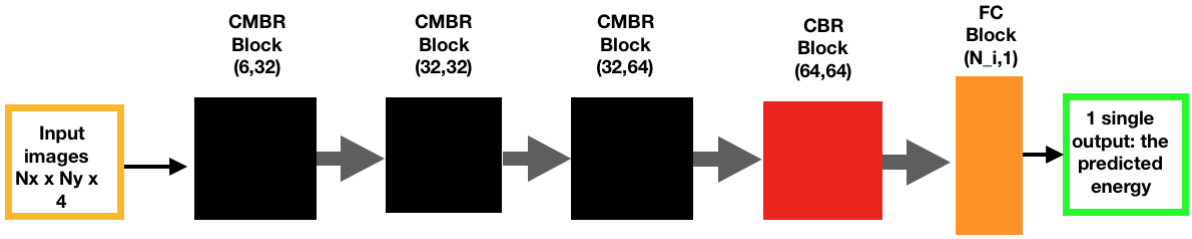
Figure A.5: *Histogram of the X position (a) and the Y position (b) of the hits with respect to the position of the barycentre of the corresponding plane for DS2.*



(a)



(b)



(c)

Figure A.6: *Different CNN architectures used during this study.*

resolution.py

```
import numpy as np

# Put the dimension of the input images
X_dimension = 86
Y_dimension = 70
Z_dimension = 6 # number of SciFi planes

# Put the caracteristics of the CNN architecture
number_of_CBR_Block = 0
number_of_CMBR_Block = 4
Z_dimension_last_block = 128

# Put the caracteristics of the CoordConv layer
k=3
S=1
P=0

#------ Start of the program -------
resolution= np.array([X_dimension,Y_dimension,Z_dimension])
print("initial resolution: ", resolution)

# action of CBR Block
for i in range(number_of_CBR_Block):
    # CoordConv layer
    resolution[0]=np.floor((resolution[0]-k+2*P)/S+1)
    resolution[1]=np.floor((resolution[1]-k+2*P)/S+1)

# action of CMBR Block
for i in range(number_of_CMBR_Block):
    # CoordConv layer
    resolution[0]=np.floor((resolution[0]-k+2*P)/S+1)
    resolution[1]=np.floor((resolution[1]-k+2*P)/S+1)
    # MaxPool layer
    resolution[0]=np.floor(resolution[0]/2)
    resolution[1]=np.floor(resolution[1]/2)

resolution[2] = Z_dimension_last_block

print("Dimension of the output of thelast convolutional block is : ",resolu
print("Number of input of the fully connected block : ",resolution[0]*resol
```

Bibliography

[1] M. Anelli et al. "A facility to Search for Hidden Particles (SHiP) at the CERN SPS" (2015). arXiv: 1504.04956 [physics.ins-det].

[2] W. Bonivento et al. "Proposal to Search for Heavy Neutral Leptons at the SPS" (2013). arXiv: 1310.1762 [hep-ex].

[3] P. Hopchev. *SciFi: A large Scintillating Fibre Tracker for LHCb*. 2017. arXiv: 1710.08325 [physics.ins-det].

[4] A. Radovic et al. "Machine learning at the energy and intensity frontiers of particle physics". *Nature* 560.7716 (2018), pp. 41–48.

[5] T. Asaka and M. Shaposhnikov. "The νMSM, dark matter and baryon asymmetry of the universe". *Phys. Lett.* B620 (2005), pp. 17–26. arXiv: hep-ph/0505013 [hep-ph].

[6] E. Graverini. "Flavour Physics with (semi)leptonic Decays at Forward Spectrometers". PhD thesis. Universität Zürich, 2018.

[7] L. Diamante. *New Portal to Unveil the Dark Sector of the Universe*. https://phys.org/news/2017-03-portal-unveil-dark-sector-universe.html. Mar. 2017.

[8] C. Savage et al. "Compatibility of DAMA/LIBRA dark matter detection with other searches". *JCAP* 0904 (2009), p. 010. arXiv: 0808.3607 [astro-ph].

[9] E. A. Bagnaschi et al. "Supersymmetric Dark Matter after LHC Run 1". *Eur. Phys. J.* C75 (2015), p. 500. arXiv: 1508.01173 [hep-ph].

[10] M. Tanabashi et al. "Review of Particle Physics". *Phys. Rev. D* 98 (3 Aug. 2018), p. 030001. URL: https://link.aps.org/doi/10.1103/PhysRevD.98.030001.

[11] G. D. Lellis. *The scattering and neutrino detector operating at LHC*. https://indico.cern.ch/event/854346/contributions/3604213/attachments/1935545/3207341/LHC_neutrinos.pdf.

[12] J. R. Ellis, K. Enqvist, and D. V. Nanopoulos. "A Very Light Gravitino in a No Scale Model". *Phys. Lett.* 147B (1984), pp. 99–102.

[13] T. Akesson et al. "Light Dark Matter eXperiment (LDMX)" (2018). arXiv: 1808.05219 [hep-ex].

[14] S. Knapen, T. Lin, and K. M. Zurek. "Light Dark Matter: Models and Constraints". *Phys. Rev.* D96.11 (2017), p. 115021. arXiv: 1709.07882 [hep-ph].

[15] *SHiP Experiment - Progress Report*. Tech. rep. Jan. 2019. URL: https://cds.cern.ch/record/2650966.

[16] C. C. Ahdida et al. "SPS Beam Dump Facility – Comprehensive Design Study" (2019). arXiv: 1912.06356 [physics.acc-ph].

[17] B. Barish. "Tau neutrino physics: an introduction". *Nuclear Physics B - Proceedings Supplements* 98.1 (2001), pp. 12–25. URL: http://www.sciencedirect.com/science/article/pii/S0920563201011902.

[18] *Physics Beyond Colliders Working Group meeting Guy Wilkinson*. https://indico.cern.ch/event/706741/contributions/3017537.

[19] *SND@LHC meeting Giovanni De Lellis*. https://indico.cern.ch/event/861997/.

[20] *18th SHiP Collaboration meeting, CERN, 30 October - 1 November 2019*. https://indico.cern.ch/event/854346/.

[21] M. Al-Turany et al. "The FairRoot framework". *J. Phys. Conf. Ser.* 396 (2012), p. 022001.

[22] R. Brun and F. Rademakers. "ROOT: An object oriented data analysis framework". *Nucl. Instrum. Meth.* A389 (1997), pp. 81–86.

[23] A. collaboration. *ALIBUILD: A simple build tool for ALICE experiment software and its externals*. https://alisw.github.io/alibuild/.

[24] S. Agostinelli et al. "GEANT4: A Simulation toolkit". *Nucl. Instrum. Meth.* A506 (2003), pp. 250–303.

[25] C. Hoppner et al. "A Novel Generic Framework for Track Fitting in Complex Detector Systems". *Nucl. Instrum. Meth.* A620 (2010), pp. 518–525. arXiv: 0911.1008 [hep-ex].

[26] T. Sjostrand, S. Mrenna, and P. Z. Skands. "A Brief Introduction to PYTHIA 8.1". *Comput. Phys. Commun.* 178 (2008), pp. 852–867. arXiv: 0710.3820 [hep-ph].

[27] T. Sjostrand, S. Mrenna, and P. Z. Skands. "PYTHIA 6.4 Physics and Manual". *JHEP* 05 (2006), p. 026. arXiv: hep-ph/0603175 [hep-ph].

[28] C. Andreopoulos et al. "The GENIE Neutrino Monte Carlo Generator". *Nucl. Instrum. Meth.* A614 (2010), pp. 87–104. arXiv: 0905.2517 [hep-ph].

[29] S. Buontempo et al. "CMS-XSEN: LHC Neutrinos at CMS. Experiment Feasibility Study" (2018). arXiv: 1804.04413 [physics.ins-det].

[30] G. de Lellis. *The scattering and neutrino detector operating at LHC*. https://indico.cern.ch/event/854346/contributions/3604213/.

[31] E. Graverini. *Dark Sector searches with proton beams*. https://indico.cern.ch/event/808335/contributions/3374022/attachments/1843389/3023688/Graverini_ESPPU_SHiP.pdf.

[32] F. Bre, J. Gimenez, and V. Fachinotti. "Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks". *Energy and Buildings* 158 (Nov. 2017).

[33] J. Loy. *Neural Network Projects with Python: The ultimate guide to using Python to explore the true power of neural networks through six projects*. 1st ed. Packt Publishing, Feb. 2019.

[34] B. van Brunt. *The calculus of variations*. Springer, Sept. 2003.

[35] C. Olah. *Calculus on Computational Graphs: Backpropagation*. https://colah.github.io/posts/2015-08-Backprop/. Aug. 2015.

[36] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[37] S. Tammina. "Transfer learning using VGG-16 with Deep Convolutional Neural Network for Classifying Images". *International Journal of Scientific and Research Publications* 9 (Oct. 2019), p. 9420.

[38] K. O'Shea and R. Nash. "An Introduction to Convolutional Neural Networks" (Nov. 2015). arXiv: 1511.08458 [cs.NE].

[39] O. M'Haimdat. *Exploring Convolutional Neural Networks (CNNs) from an iOS Developer's Perspective*.

[40] R. Liu et al. "An Intriguing Failing of Convolutional Neural Networks and the CoordConv Solution". *arXiv* (2018). arXiv: 1807.03247 [cs.CV].

[41] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[42] *Original algorithm from S. Shirobokov*. https://github.com/shir994/SND_trackers.

[43] X. Jia et al. "Highly Scalable Deep Learning Training System with Mixed-Precision: Training ImageNet in Four Minutes". *arXiv* (2018). arXiv: 1807.11205 [cs.LG].

[44] L. N. Smith. "No More Pesky Learning Rate Guessing Games". *CoRR* abs/1506.01186 (2015). arXiv: 1506.01186. URL: http://arxiv.org/abs/1506.01186.

[45] *Détecteurs de particules*. Monographies de Cambridge sur la physique des particules, la physique nucléaire, et cosmologie. Cambridge. URL: https://cds.cern.ch/record/306826.

[46] R. Wigmans and R. Wigmans. *Calorimetry: Energy Measurement in Particle Physics*. International series of monographs on physics. Clarendon Press, 2000. URL: https://books.google.ch/books?id=vD9RFluMD5sC.