University of Genova

School of Mathematical, Physical and Natural Science

Physics Department

Università di Genova

Master's Degree in Physics

A novel approach for real-time identification of hadronic final states at the High-Luminosity LHC.

Candidate:
Alessandro Zaio

Supervisors:
Dr. Andrea Coccaro
Prof. Carlo Schiavi

Co-supervisor:
Prof. Antonino Sergi

October 18, 2023

Contents

Introduction

The trigger system is a fundamental component of any collider experiment since it is responsible for deciding whether data collected from a given bunch-crossing interaction will be saved or not for later study. This is especially true for the experiments at the Large Hadron Collider (LHC), where the bunches of protons are separated by 25 ns, corresponding to a frequency of 40 MHz. Each bunch-crossing interaction produces multiple collisions, with the secondary ones called pile-up. For example, Run 2 had a peak instantaneous luminosity of $\mathcal{L} = 2 \times 10^{34}$ cm^{-2} s^{-1}, corresponding to an average number of pile-up collisions in 2018 of $\langle \mu \rangle = 36.1$.
The trigger of the ATLAS experiment consists of an hardware trigger called Level 1 (L1), followed by a software trigger called High Level Trigger (HLT). L1 reduces the rate of the events to 100 kHz and the HLT operates a further reduction, selecting events for permanent storage and offline processing with an output rate on the order of 1 kHz. While L1 only uses data from the calorimeters and the muon spectrometer detector, the HLT also relies on data coming from the inner tracking detector.
The L1 trigger, beyond selecting the collisions of interest, also called events, indicates the so-called Regions of Interest (RoIs) with the HLT focusing on these regions for reconstructing tracks. The track reconstruction at the HLT is divided into Fast Tracking and Precision Tracking: the first one prioritizes efficiency over purity and provides a fast reconstruction of the tracks in order to keep up with the required latency. The reconstructed candidate tracks then go through a further trigger selection with a lower rate, allowing for finer tracking algorithms to process the data.
Precision tracks can then be used for the b-tagging algorithms, that try to answer the question of whether a jet comes from a b-quark or not, since we are very much interested in such physical processes, like for example the Higgs boson decaying into two b-quarks ($H \to bb$).
When a charged particle goes through a pixel layer, a charge deposit is usually registered in multiple pixels, which are grouped together to form a cluster. From a cluster a space point is computed with an average over the positions of the activated pixels, weighted by the charge deposited in each one.
The processing done in the Fast Tracking uses combinatorial algorithms, and looks for triplets of space points that could belong to the same track; this kind of algorithms will require more time when engaging with more dense environments, since there are more combination available to look for.

Observing the processing time per RoI as a function of the pile-up, we can see a rapid increase of the CPU computing time; this becomes a problem if we try to apply the same algorithms to events with a very high density of tracks.
After Run 3, which is ongoing, the LHC will undergo a series of updates that could lead to an integrated luminosity expected for Run 4 of 3000 fb^{-1}, which will provide a huge increase, if we compare it with the 150 fb^{-1} that should be obtained from Run 3.
The new version of the collider will be called High Luminosity-HLC (HL-HLC), and is prospected to provide events with an average pile-up of 140 to 200, presenting a very challenging environment for the present experiments at the LHC.
With this in mind, the current trigger used in ATLAS won't be able to process the amount of data for such high densities, while keeping the same event selection rates at which it is performing today. This limit is connected to the maximum computing power available for the CPU farms that handle the trigger operations.
In order to use the present trigger for the HL-LHC environment, it would be necessary to raise the energy thresholds for the selection, in order to reduce the amount of data to be transfered, but this involves loosing interesting events for further processing.
The need for an update of the Trigger an Data Acquisition (TDAQ) System becomes evident, and in this work we propose a prototype for a Machine Learning (ML) algorithm that could perform Fast Tracking without the heavy dependence on pile-up that the present combinatorial algorithms show, allowing to process HL-LHC-like events without raising the energy thresholds.
Raising the thresholds would have a strong impact on the ATLAS physics reach, including the search for double-Higgs production in the most probable decay channel, $HH \to 4b$. This process, and the triggering strategies for preserving it in the saved data, is studied in detail in the context of this thesis.
The production of two Higgs bosons in the same events is extremely rare but it is prospected to be discovered at the HL-HLC. A Higgs boson can decay into HH, since in the Standard Model the trilinear Higgs self-coupling is allowed, but we'll also explore the possibility of HH resonant production from particles beyond SM.
We'll be looking at Montecarlo simulated events of $HH \to 4b$, and we're interested to see if lowering the momentum and energy thresholds of the HLT could lead to a better acceptance of the HH events over the background from QCD events. Such a reduction of the thresholds is possible only if the triggers perform faster, so we wonder how much faster the triggers need to be.
After providing an introduction to some ML concepts, we will discuss the proposed ML-based tracking algorithm: since we are at the first steps in this direction, we train and apply the algorithm on a set of toy events that we build from scratch, simulating tracks passing through a simplified model for the inner tracking system. Once we have properly analyzed the behaviour of the algorithm on such toy events, we'll be ready to apply it on ATLAS events, but this is left for further developments.
First of all, we would like the algorithm to be able to distinguish hits generated by tracks from random noise that we add to the events, so that the algorithm performs a denoising

task. The noise on the layers of the simulated detector will be modeled on the density of hits found in actual $t\bar{t}$ jets, since it is a future prospect of this work to apply the algorithm to this category of jets.
Our algorithm will be a Convolutional Neural Network (CNN), which is designed to receive images as inputs. We'll justify the choice of this kind of neural network, because while CNNs architecture can be easily implemented and accelerated on FPGA devices, it becomes very complicated for more sophisticated and better performing algorithms. We will then study the performance of some CNN models on the toy events that we are producing.
We will then study the robustness of the prototype algorithm as a function of tracks kinematics and of the pile-up noise level.
In conclusion, based on the currently available results, we'll also present two other ideas that can be achieved with similar ML-based algorithms, counting how many signal tracks are contained in an events and inferring the p_T value for single-track events.
We will also briefly explain how acceleration techniques suitable for the proposed architecture could make the inference faster, and how we are planning to change our CNN in order for it to evolve from a denoising algorithm to actually performing full tracking.

Chapter 1

Physics with the ATLAS detector

In this first chapter I present an overview of the physics studied at the ATLAS experiment, starting with the theoretical framework of particle physics, the Standard Model (SM). I'll briefly describe the properties of the SM as a Quantum Field Theory (QFT), discussing the symmetries and the particle content of the theory. I'll later talk about the role of the Quantum Chromo Dynamics (QCD), defining some fundamental concepts such as asymptotic freedom and colour confinement. Next up I'll present the Electroweak interaction and introduce the Higgs boson, the only scalar elementary particle in the SM. I will use the tools listed thus far to define particle jets, which are of central importance for high-energy particle physics; in particular, the jets formed after the production of b-quarks, also called b-jets, have a key role in the physics treated in this work, and in this context I'll also briefly explain what we mean with b-tagging and why it is important for the physical searches in ATLAS.
After this rapid overview of high-energy particle physics, I'll present the Large Hadron Collider (LHC), the largest particle accelerator and collider in the world, able to reach energies at the TeV scale. In correspondence of one of the four points of collision of the hadron beams, we have the ATLAS experiment; I'll present the structure of the ATLAS detector, explaining the role that each sub-detector has in the reconstruction of the physical processes.
Inside ATLAS a collision happens every 25 ns, so that it's clearly not possible to store the data for every event; it becomes necessary to discard the great majority of these events, choosing to save only the ones containing physically interesting processes. This task is entrusted to the Trigger system, which decides if the event has to be saved for offline analysis or if the data can be thrown away. The trigger system is divided into two parts: the first applies a hardware-based selection, while the second one is software-based and is called High Level Trigger (HLT). The main topic of my thesis work will focus on the real-time reconstruction of tracks, also called tracking, which is introduced in this chapter and then analyzed in more detail in chapter 3.

1.1 The Standard Model

The SM of particle physics is the theory describing the known elementary particles and their interaction through the fundamental forces, the electromagnetic, strong and weak forces, with the exception of gravity, which is not included in the theory.
The main contribution to the theory are attributed to Sheldon Glashow, who managed to combine the electromagnetic and the weak interactions in a single framework in 1961, and to Steven Weinberg and Abdus Salam who added in 1967 the Higgs Mechanism to Glashow's theory.
The theory has been tested again and again over the last decades, showing astonishing precision in predicting the results of the experiments.
The SM is a non-abelian gauge theory, which is invariant under the gauge Group:

$$G = SU(3)_C \otimes SU(2)_L \otimes U(1)_Y. \tag{1.1}$$

In the following I will show the particle content of the SM, and also provide a description of the fundamental forces within the model, from the point of view of Quantum Field Theory (QFT). This introduction is mainly inspired by [1] and [2].

1.1.1 Particle Content of the SM

The SM contains all the known elementary particles, which consist of twelve spin-$\frac{1}{2}$ fermions, twelve spin-1 vector bosons and one single scalar particle that we can se displayed in figure 1.1.
The fermion content is organized in 3 generations, with each one having one charged lepton, a neutrino and two quarks The particles of different generations have the same quantum numbers with the exception of the mass, which grows for each generation; this does not concern the neutrinos, which are massless in the SM. Furthermore to each fermion in the model is associated an anti-particle with same mass but opposite charges, even though nature seems to favour particles over anti-particles, since ordinary matter in the universe is only composed of first generation particles.
Particles in the SM may carry three types of charge, each associated with an interaction, which are the electric charge, the colour charge and the weak charge. Quarks carry both electric and colour charge and for each generation there is an "up-like" and a "down-like" quark with electric charges $\frac{2}{3}$ and $-\frac{1}{3}$. As we'll see later in more detail, coloured states do not propagate freely because of the property of the strong interaction called colour confinement, according to which free states can only be colour-less. This property also involves quarks, which can only be found in bound colour-less states called hadrons, which are classified in baryons, formed by three quarks, and mesons, formed by a quark and an anti-quark.
Neutrinos instead only carry the weak charge, hence they can only interact weakly; indeed every particle within the SM, except for the photon and the gluons, carries a weak charge.
The charged leptons (e, μ, τ) carry an electric charge of -1 and for each there is an associated neutrino, which are ν_e, ν_μ and ν_τ.

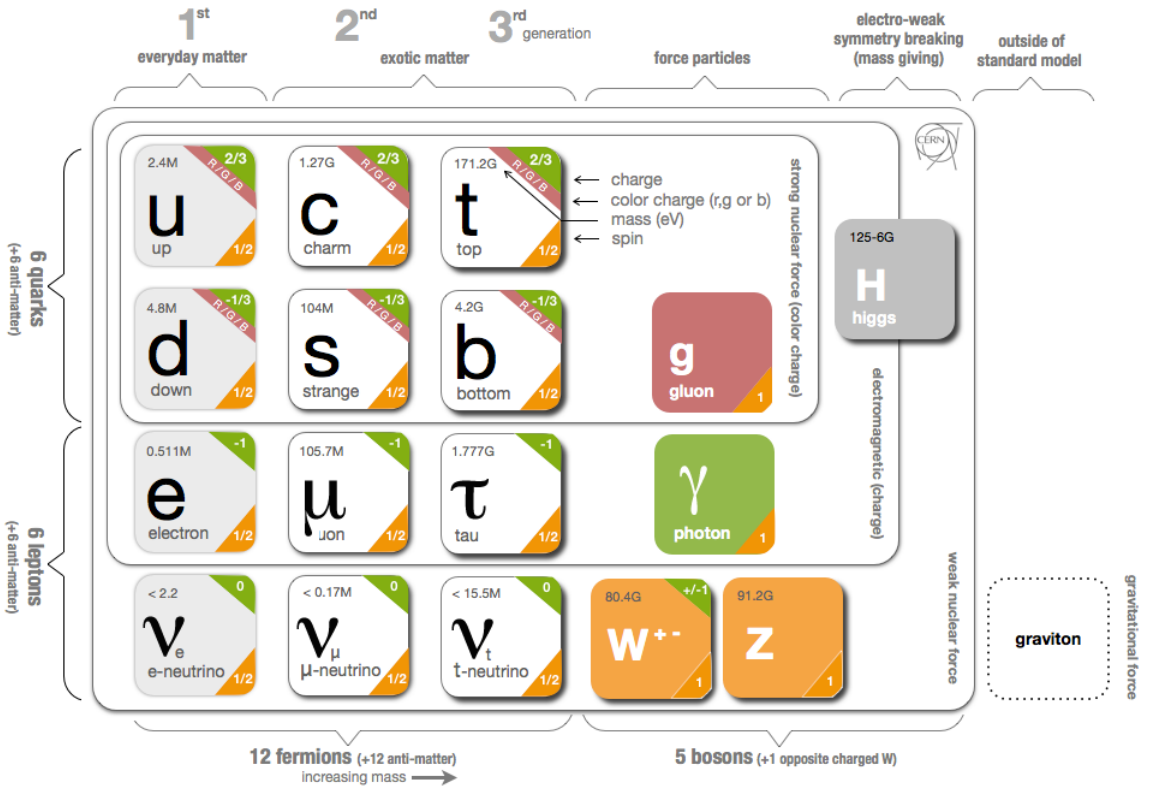


Figure 1.1: Display of the particle content within the SM, with each mass, electric charge, colour charge and spin indicated.

The vector bosons are gauge bosons, which have the task of mediating the interactions between the particles; within the gauge bosons we have the photon γ, mediating the the electromagnetic force, the massive Z and $W^{\pm}$ associated to the weak force, or more precisely the electroweak force and also eight types of gluons g, which carry the strong force.
Finally, the Higgs boson is the only scalar particle of the SM, carrying only the weak charge. The Higgs H is of primary importance within the SM, because of the Higgs mechanism and the electroweak symmetry breaking, that I'll both describe later in this chapter.

1.1.2 QED and QCD

We now try to get more into the theoretical framework of the SM, briefly describing the electromagnetic and strong interactions.
QFT is the result of the adaptation of classical quantum mechanics to the laws of special relativity, which has the result of requiring that the particles have to be represented as fields.
Quantum Electrodynamics (QED) is the description of the electromagnetic interaction in a quantum-relativistic framework, which is obtained requiring that the Lagrangian describing the interaction is invariant under a local $U(1)$ gauge transformation, described by:

$$\psi(x) \to e^{-i\alpha(x)}\psi(x), \quad A_\mu(x) \to A_\mu(x) - \frac{1}{\mathrm{e}}\,\partial_\mu\alpha(x), \tag{1.2}$$

where ψ(x) is the field function of a massive spin-$\frac{1}{2}$ fermion, A_μ is a gauge vector field, which corresponds to the photon, and e is the elementary electric charge.
The resulting Lagrangian $\mathcal{L}_{QED}$ which is invariant under the $U(1)$ gauge transformation is:

$$\mathcal{L}_{\mathrm{QED}} = \mathcal{L}_{\mathrm{Dirac}} + \mathcal{L}_{\mathrm{Maxwell}} + \mathcal{L}_{\mathrm{interaction}} \tag{1.3}$$

$$= \bar{\psi}(i\partial\!\!\!/ - m)\psi - \frac{1}{4}\,F^{\mu\nu}F_{\mu\nu} - \mathrm{e}\,\bar{\psi}\gamma^\mu\psi A_\mu = \bar{\psi}(iD\!\!\!/ - m)\psi - \frac{1}{4}\,F^{\mu\nu}F_{\mu\nu}, \tag{1.4}$$

where $\partial\!\!\!/ = \partial_\mu\gamma^\mu$ is the contraction between the partial derivative and the Dirac matrices γ^μ, $F_{\mu\nu} = \partial_\mu A_\nu - \partial_\nu A_\mu$ is the electromagnetic field tensor and $D_\mu = \partial_\mu + ie\,A_\mu$ is the covariant derivative.
While the $\mathcal{L}_{\mathrm{Maxwell}}$ term is invariant under the gauge transformation, the term describing a free propagating fermion $\mathcal{L}_{\mathrm{Dirac}}$ is not invariant, hence the need of the introduction of the $A_\mu(x)$ field to restore the invariance. From the lagrangian $\mathcal{L}_{\mathrm{QED}}$ the Feynman diagrams describing the propagator of the spin-$\frac{1}{2}$ fermion, the propagator of the photon and the interaction between a photon and a fermion and an anti-fermion can be derived.
The invariance of the Lagrangian under $U(1)$ implies, under Noether's theorem, the conservation of the electric charge. Furthermore, due to the commutative nature of the $U(1)$ group, there is no interaction involving multiple photons; if we instead consider a non-abelian group, we'll observe a different phenomenology.
The strong interaction is instead descrived by Quantum Chromodynamic (QCD), which is a non-abelian gauge theory based on the SU(3) group. In this case we consider the quark field ψ_i with $i \in \{1, 2, 3\}$, since SU(3) lives in a space of dimension 3, where with each index indicates a different colour, and so the gauge transformation acts on the colour space.
In this case the gauge transformation take the form:

$$\psi_i(x) \to e^{-i\alpha_a(x)t_a}\psi_i(x), \quad A^a_\mu(x) \to e^{i\alpha_b(x)t_b}\left(A^a_\mu(x) + \frac{i}{g_S}\,\partial_\mu\alpha(x)\right)e^{-i\alpha_b(x)t_b}, \tag{1.5}$$

$$\text{with} \quad t_a = \frac{1}{2}\,\lambda_a, \quad [\lambda_a, \lambda_b] = i\,f^c_{a\,b}\,\lambda_c. \tag{1.6}$$

In equation 1.5 the sum over the indexes $a, b \in \{1, ..., 8\}$ is implied and the matrices t_a are the generators of the algebra associated to $SU(3)$. This time the eight fields A^a_μ represent the gluons, and g_S is the strong coupling strength. The generators t_a can also be written in terms of the Gell-Mann matrices λ_a, as we see in equation 1.6, and since the $SU(3)$ group is non-abelian, the commutators of the Gell-Mann matrices is different from zero, and is expressed with the completely anti-symmetric structure constant $f^c_{a\,b}$.
In analogy to what we have seen for QED we obtain the following lagrangian:

$$\mathcal{L}_{\text{QCD}} = \bar{\psi}_i \left(i \not{D} - m_i\right) \psi_i - \frac{1}{4} F^{a\,\mu\nu} F^a_{\mu\nu}, \tag{1.7}$$

$$\text{with} \quad F^a_{\mu\,\nu} = \partial_\mu A^a_\nu - \partial_\nu A^a_\mu + g_S\, f^a_{b\,c} A^b_\mu A^c_\mu. \tag{1.8}$$

In equation 1.7 we have again introduced the covariant derivative for the strong force, defined as $D_\mu = \partial_\mu - i\, g_S\, \frac{\lambda_a}{2}\, A^a_\mu$.
The strength tensors $F^a_{\mu\,\nu}$ this time includes also a term quadratic in the gluon fields, which is necessary to ensure the invariance of the lagrangian, and is connected to the non-abelian nature of the SU(3) group.
Building the Feynman diagrams for the strong force, the presence of the quadratic term causes diagrams describing the self-coupling for the gluons, involving the self-interaction of three gluons (first order in g_S) and also four gluons (second order in g_S).
We now introduce the coupling constant for the strong interaction:

$$\alpha_S = \frac{g_S^2}{4\pi}, \tag{1.9}$$

which describes the strength of the interaction. The coupling constant actually depends on the energy of the of the interaction; indeed in the context of renormalization, an arbitrary energy scale μ is introduced, at which we observe the dependence of α_S with respect of the energy of the interaction Q. Without going into further details we report here the result obtained for α_S form the perturbative computation considering one single loop:

$$\alpha_S(Q^2) = \frac{\alpha_S(\mu^2)}{1 + \alpha_S(\mu^2)\, \beta_0 \ln\left(\frac{Q^2}{\mu^2}\right)} \tag{1.10}$$

$$\text{with} \quad \beta_0 = \frac{11\, c_A - 2\, n_f}{12\pi}. \tag{1.11}$$

But since c_A equals the dimension of the representation, so $c_A = 3$, and n_f is the number of flavours of quarks which are lighter than μ, which has the maximum value of $n_f = 6$, we obtain that β_0 is negative and so the coupling constant decreases at higher energies. We also notice a value for μ at which the denominator in 1.10 becomes zero; this value is referred to $\alpha_S(m_Z)$, with $m_Z = 91.18$ GeV and takes the name of Landau's Pole, corresponding to $\Lambda_{QCD} \simeq 200$ MeV. This divergence signifies the failure of the perturbative approach, since perturbation theory can be applied when $\alpha_S << 1$, and consequently when $Q >> \Lambda_{QCD}$.

This implies that at large distances, which correspond to low energy scales, the coupling between two particles with colour charge increases, and this justifies the fact that coloured states cannot be observed directly; this property takes the name of colour confinement. On the other hand for small distances, corresponding to high energy scales, the coupling strength decreases, causing the effect called asymptotic freedom.

1.1.3 Electroweak Unification and Higgs

The electromagnetic and weak interactions can be unified into one single theory, which is based on the non-abelian gauge group $SU(2)_L \otimes U(1)_Y$ and is associated to the following lagrangian:

$$\mathcal{L}_{\text{EW}} = \mathcal{L}_{\text{gauge}} + \mathcal{L}_{\text{fermion}} + \mathcal{L}_{\text{Higgs}} + \mathcal{L}_{\text{Yukawa}}. \tag{1.12}$$

In the following I'll go through each term of the lagrangian 1.12 and their physical meaning. From the observation of the Beta decay, it is derived that the weak interaction involves only left-handed fermions and right-handed anti-fermions, where right-handed and left-handed states are eigenstates of the chirality operator with eigenvalues ± 1.
We now see how this phenomenon is reflected in the theory: left-handed fermion fields transform according to the fundamental representation of $SU(2)$, forming doublets such as:

$$\begin{pmatrix} \nu_e \\ e \end{pmatrix}_L, \quad \begin{pmatrix} u \\ d \end{pmatrix}_L \tag{1.13}$$

and analogously for the other generations.
On the other hand right-handed fermions are representation of the singlet of $SU(2)_L$, and so they only transform according to $U(1)_Y$.
In the same way as we have already done for the $U(1)$ and the $SU(3)$ groups, we introduce three vector fields W^a_μ, with $a \in \{1, 2, 3\}$, and another vector field B_μ. We have three different W^a_μ because there are three generators of the algebra associated to $SU(2)$, namely the Dirac matrices σ^a.
Introducing the left-handed field ψ_L, a $SU(2)$ doublet, and the right-handed field ψ_R, which is a singlet, the gauge transformations take the following form:

$$\psi_L \to e^{i\, g_2\, \alpha_a(x) \frac{\sigma_a}{2} + i\, g_1\, \frac{Y}{2} \theta(x)}\, \psi_L, \tag{1.14}$$

$$\psi_R \to e^{i\, g_1\, \frac{Y}{2} \theta(x)}\, \psi_R, \tag{1.15}$$

where g_1 and g_2 are the coupling constants associated to the two gauge groups, and Y is the weak hypercharge.
As we did before, we introduce the field strength tensors $W^a_{\mu\nu}$ and $B_{\mu\nu}$ associated to the gauge bosons and write:

$$\mathcal{L}_{\text{gauge}} = -\frac{1}{4}\, W^a_{\mu\nu} W_{a\,\mu\nu} - \frac{1}{4}\, B_{\mu\nu} B_{\mu\nu} \tag{1.16}$$

At this stage introducing the mass terms for the gauge bosons, of the form $m_W^2 W_\mu W^\mu$, would violate the gauge invariance. We'll later see that the spontaneous breaking of the electroweak symmetry will allow us to introduce these mass terms.
Moving on the fermion term in the lagrangian is:

$$\mathcal{L}_{\text{fermion}} = \bar{L}_L^i \, i \not{D}_\mu^L \, L_L^i + \bar{Q}_L^i \, i \not{D}_\mu^L \, Q_L^i + \bar{l}_R^i \, i \not{D}_\mu^R \, l_R^i + \bar{u}_R^i \, i \not{D}_\mu^R \, u_R^i + \bar{d}_R^i \, i \not{D}_\mu^R \, d_R^i, \tag{1.17}$$

where the index i denotes the i-th generation and the sum over $i \in \{1, 2, 3\}$ is implied. Furthermore L_L^i and Q_L^i denote the doublets for each generation, such as the ones in equation 1.13 and l_R^i, u_R^i and d_R^i denote the lepton, the up-like quark and the down-like quark of the i-th generation.
We now introduce the Higgs boson, which is a $SU(2)_L$ doublet of complex scalar fields:

$$\Phi(x) = \frac{1}{\sqrt{2}} \begin{pmatrix} \varphi^+(x) \\ \varphi^0(x) \end{pmatrix}. \tag{1.18}$$

We can now write the Higgs term of the lagrangian in equation 1.12:

$$\mathcal{L}_{\text{Higgs}} = (D_\mu \Phi)^\dagger \, D^\mu \Phi - V(\Phi) \tag{1.19}$$

where D_μ is the covariant derivative and $V(\Phi)$ the Higgs potential. In chapter 2 we'll see in more detail the Higgs mechanism, in which the shape of the Higgs potential causes the spontaneous breaking of the $SU(2)_L \otimes U(1)_Y$ symmetry , of which only the subgroup $U(1)_{EM}$ remains respected.
The Higgs mechanism also justifies the masses of the $W^\pm$ and Z bosons, indeed, applying a change of basis, it's possible to rewrite the fields $W_\mu^{1,2,3}$ and B_μ as the physical fields W_μ^+, W_μ^-, Z_μ and A_μ, which describes the photon.
The $W^\pm$ and Z bosons obtain a mass term via the Yukawa couplings, as well as the other massive fermions e have listed. The Yukawa term in the electroweak lagrangian is:

$$\mathcal{L}_{\text{Yukawa}} = -\Gamma_{ij}^l \, \bar{L}_L^i \, \Phi \, l_R^i - \Gamma_{ij}^d \, \bar{Q}_L^i \, \Phi \, d_R^i - \Gamma_{ij}^u \, \bar{Q}_L^i \, \Phi^c \, u_R^i + \text{h.c.}, \tag{1.20}$$

where Φ^c is the charge conjugate of Φ, h.c. stands for hermitian conjugate and the Yukawa couplings $\Gamma_{ij}^{l,d,u}$ are free parameters of the SM. In the Yukawa term we notice the presence of the Higgs field Φ, which contributes to the mass terms of the massive particles within the SM.

1.2 Jets

When studying the QCD we encountered the concept of colour confinement, stating that at the strong coupling increases for lower energies, so that we obtain $\alpha_S >> 1$ for energy scales below 1 GeV. This phenomenon implies that coloured states cannot be observed directly because they'll always generate and combine with other colour charged objects, forming hadrons which will be found in the final state.
In the following I want to study in more detail this process, for which it will be necessary the introduction of the concept of jet. We'll also talk about vertexing algorithms and b-tagging, which play an important role in the reconstruction and in the analysis of jets.

1.2.1 Definition of a Jet

In the hard scattering between two hadrons, for example protons, the structure of the hadrons breaks apart revealing the inner structure, which is composed of quarks and gluons, collectively called partons. The process where the scattering between two hadron causes the generation of partons, is called fragmentation.
Due to the high energy of the interaction, the partons in the final state will usually be in the energy range of asymptotical freedom; these partons will then propagate and emit soft and collinear gluons, which in turn emit other gluons, generating what is defined as a parton shower.
The emission of gluons causes the decrease of the energy of the partons, that will eventually fall below the energy scale of asymptotical freedom, starting the non-perturbative process of hadronization, which is a direct consequence of colour confinement and causes the generation of hadrons in the final state.
A schematization of the process that I have just described is found in figure 1.2.

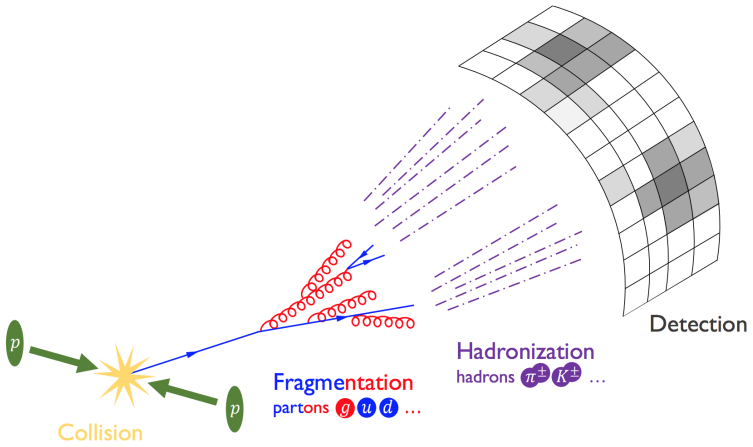


Figure 1.2: Illustration of the production of jets, with the fragmentation, the emission of soft gluons and then the hadronization.

What is empirically observed is a stream of collimated particles, which takes the name of jet and includes hadrons, photons, electrons and muons.
The study of jets is fundamental for high energy particle physics, in order to interpret the data obtained by the detector and reconstruct the physical process.
As we'll see in this thesis, many interesting physical process have jets in their final states, and so it is necessary to develop techniques to deal with this objects.
First of all it is necessary to define a jet given the ensemble of particles in the final state, but this definition is not unique and depends on the clustering algorithm used.
The most common algorithm for jet reconstruction is the anti-k_t algorithm [3], which enacts a clustering of the four-momenta of the particles into a cone-shaped object, as we can see in figure 1.3.

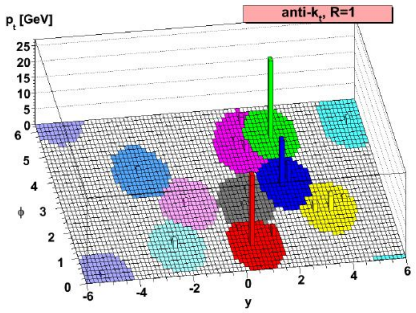


Figure 1.3: Illustration of jets reconstructed with the anti-k_t algorithm.

1.2.2 Vertexing

The vertex is the point were the interaction between particles occures, and is also the origin of the tracks that will be observed by the detector. The Primary Vertex (PV) is of peculiar importance, since it's the point where the interaction between the colliding hadrons took place, and a precise estimate for the PV is of key importance to correctly reconstruct the event detected. Also the secondary and tertiary vertices, where the decay of the products of the PV occur, are important to be reconstructed, as they play a role in heavy-flavour tagging.
The PV are found with an algorithm, which starts with a set of tracks satisfying a selection criteria, and then iteratively computes the best estimate for the vertex, removing at each step the tracks which are less compatible with it, and computing again the position of the PV. Later in the thesis we'll encounter vertexing algorithms in the context of the real-time reconstruction of tracks.

1.2.3 b-tagging

The identification of heavy-flavour jets is of major importance in particle-physics analyses, because several interesting physical processes, such as $H \to b\bar{b}$ or $H \to c\bar{c}$, have heavy flavour quarks in their final state.
Flavour tagging selects the signal-like jets and discards background-like jets, and is used both for the real-time selection of events and for the analysis of offline data.
A b- or c-jet is identified by observing the b- or c-hadrons within the jet itself, rather than directly observing the intiating quarks. Multiple signatures of b- and c-hadrons can help to distinguish the associated jets from light-flavoured jets; first of all, b-hadrons have a lifetime of around 1.5 ps [4] which corresponds to a travelled distance of about 2.5 mm for a momentum of 30 GeV. Also the decay multiplicity is an helpful variable, since B^0 hadrons have an average of 5 stable particles as the product of the decay.
A way to parametrize the displacement of the secondary vertex with respect to the PV is to use the Impact Parameters (IPs) of the associated tracks, where the transverse IP d_0 is the minimum distance between the track and the axis of the beam-pipe, which we'll properly define in the following chapter, and the longitudinal IP $z_0 \cdot \sin(\theta)$ is the analogous in the longitudinal plane, as we can see in figure 1.4.

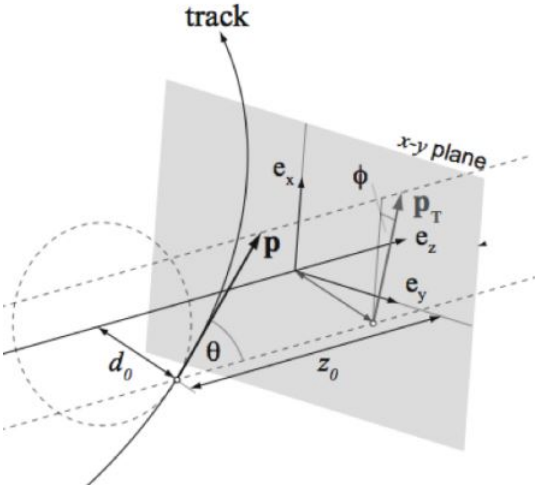


Figure 1.4: Illustration of the transverse and longitudinal impact parameters.

A positive transverse IP d_0 is consistent with a track being generated from a secondary vertex which comes after the PV along the trajectory of the particle being produced at the PV, if the track is instead consistent with a secondary vertex which precedes the PV, then d_0 is negative.
If we consider c-hadrons, they have similar properties as b-hadrons, which makes it hard to distinguish the two. Anyway since c-hadrons have a shorter lifetime, a lower decay multiplicity and also smaller d_0 values it is still possible to discriminate between these two flavours.

For light-flavour jets, most of the tracks are produced at the PV, so it is way easier to tell them apart from b and c-hadrons.
In figure 1.5 we can observe the distribution of the significance of the transverse IP $s_{d_0} = \frac{d_0}{\sigma(d_0)}$ and the longitudinal IP $s_{z_0} = \frac{z_0 \cdot \sin\theta}{\sigma(z_0 \cdot \sin(\theta))}$, for the tracks in b, c and light-flavour jets [5].

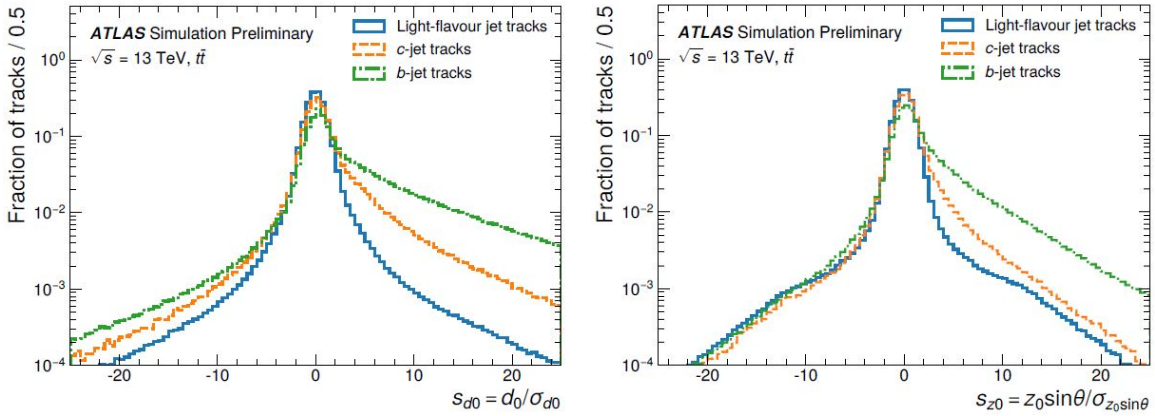


Figure 1.5: Distribution of s_{d_0} (left) and s_{z_0} (right), for the tracks in b, c and light-flavour jets

For light-flavour jets we can clearly notice that the distribution of the significance is basically symmetric for both d_0 and z_0; this is because the distance travelled by light-hadrons produced at the PV is in the order of the μm and so the distributions of the IPs will be centered in zero and show a symmetry for positive and negative values simply due to resolution effects. For c and b hadrons instead, since the distances traveled by the associated hadrons is instead in the order of the mm, there is a noticeable asymmetry towards positive values of the IPs, due to their larger average lifetimes.
A b-tagging algorithms receives different variables related to the tracks within the jet, and then the output consists in three values (p_b, p_c, p_u), which correspond to the probabilities that the input jet is a b, c or light-jet.
Within the ATLAS collaboration many baseline b-tagging algorithms have been developed using different approaches, which are: IP-based, vertex-based and soft-muon-based. There are also high-level taggers that utilize the outputs of the baseline algorithms and with techniques such as Boosted Decision Trees (BDT) or deep-learning neural networks provide more accurate values for (p_b, p_c, p_u).
More recently there are also algorithms which receive as input the tracks, and using more refined machine learning architecture directly compute the flavour probabilities; for example the DIPS algorithm [6] [7] is based on Deep-Sets, while GN1 is based on Graph Neural Networks (GNN). In chapter 3 we'll see an application of a version of DIPS adapted to work at the trigger level, which is called fastDIPS, and plays an important role in the real-time track reconstruction for Run 3.

1.3 The Large Hadron Collider

In order to test the physics of the SM, a large and complex machinery is required, that could reach energies of the TeV scale, hence allowing to explore the length scales of subatomic particles, in the order of less than 10^{-15}m. Such a technology is present at the Large Hadron Collider (LHC) [8], at the European Organization for Nuclear Research (CERN) located near Geneva, Switzerland.
The LHC is a two-ring superconducting hadron accelerator and collider, located 100 m below ground, and with its 26.7 km circumference is the largest and most energetic particle accelerator ever built. LHC is normally used to collide protons with protons (pp), but it can also run heavy ion collisions, like proton-lead or lead-lead.
The operations at LHC are organized with an alternation of Runs and long shutdowns; during the Runs the beams are circulating in the pipe, collisions are produced and the relative data is stored, while with the shutdowns the beam is stopped and a process of technological upgrade of the components of LHC takes place, in order to maintain and enhance the features of the beam and the detectors for the next Run.
The acceleration and collision of protons started in 2009 with Run 1(2009-2013) using beams with a center-of-mass energy of $\sqrt{s}$=7 and 8 TeV. In 2013 the first Long Shutdown took place, upgrading multiple aspects of the LHC, such as the accelerating apparatus, enabling collisions at $\sqrt{s}$=13 TeV for Run 2(2015-2018). After the second Long Shutdown, Run 3 started in April 2022, with an energy of $\sqrt{s}$=13.6 TeV, and is currently ongoing.

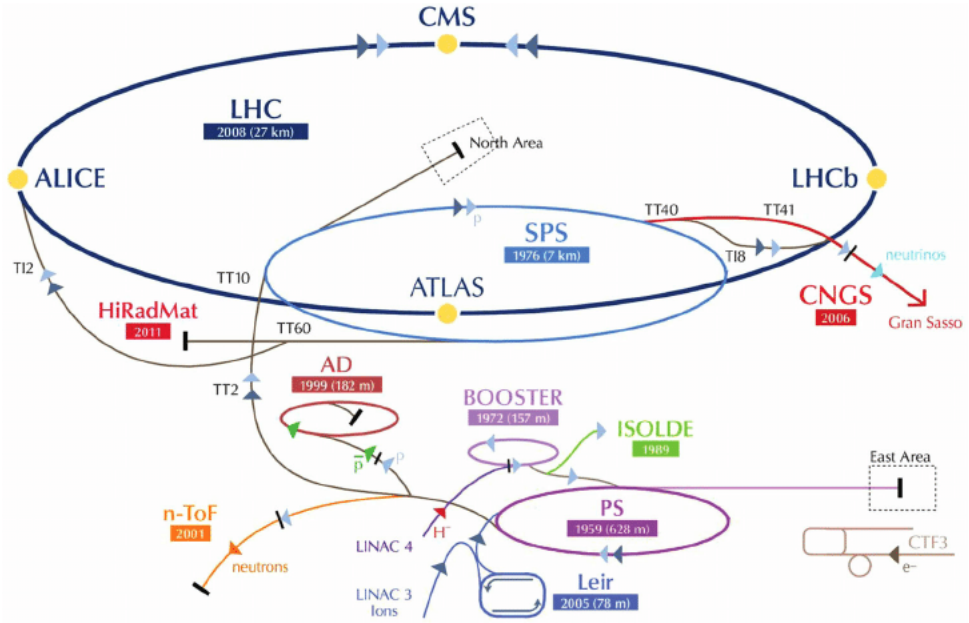


Figure 1.6: Diagram of the CERN accelerator chain: the protons start from LINAC4, go through the PS and SPS, and are injected into the LHC.

As we can see in figure 1.6, the LHC is the final element of a large accelerator chain; the starting point is the production of the protons, which are obtained from an ionized hydrogen gas and then injected inside the linear accelerator (LINAC 2 in Run 2 and LINAC 4 in Run 3), where they are accelerated up to 50 MeV. The following steps are the Booster, accelerating to 1.4 GeV, the Proton Synchrotron (PS), reaching 25 GeV and the Super Proton Synchrotron (SPS), which injects the protons with an energy of 450 GeV into LHC. Here, superconducting magnets guide the two oppositely moving beams along the circular path into separated pipes, and a series of resonant frequency cavities accelerate the protons up to 6.8 TeV.
The beams of protons are organized in bunches, typically 2400, each one separated from the next one by 25 ns, and containing $\mathcal{O}(10^{11})$ protons. The bunches, traveling in the two parallel pipes, are brought to collision in four crossing points, each hosting an experiment, with a center of mass of $\sqrt{s}$=13.6 TeV for Run 3. The products of the collision of each bunch, as recorded by the experiment, are collectively called events.
Of the four experiments hosted at the LHC, two of them, ATLAS and CMS, are multi-purpose experiments interested in SM precision measurements and searches beyond the SM.
The LHCb experiment focuses on c and b-hadrons with large emphasis on CP-violating processes, while the ALICE experiment focuses on heavy-ion collisions.
Since a huge amount of data is produced at LHC, a collaborative effort is necessary to analyze all the events, so that a distributed computing and data storage infrastructure is employed, called the Worldwide LHC Computing Grid or WLCG. Indeed from the Tier-0 center, the data is distributed in seven Tier-1 facilities all around the world, where the events are reconstructed and analyzed.
The processed events are then sent to Tier-2 centers, where the data is stored in a lighter and more accessible format, ready for the physical analysis.

1.3.1 Luminosity and Pile-up

Apart from the center of mass energy in the collisions, also the instantaneous luminosity is an important feature of a particle collider. The instantaneous luminosity is defined as the number of particles per unit area, per unit time, and per unit solid angle that are available for the collisions, and is expressed in $\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$.
Considering a collider with a Gaussian effective beam area $\mathcal{A} = 4\pi\sigma_x\sigma_y$, where σ_x and σ_y are the widths of the beam in the directions perpendicular to the beam, the instantaneous luminosity is:

$$\mathcal{L} = f_{\mathrm{rev}} \frac{N_1 N_2}{4\pi\sigma_x\sigma_y} F(\theta_{\mathrm{c}}) \tag{1.21}$$

with f_{rev} the revolution frequency, $N_{1,2}$ the number of protons in each beam and $F(\theta_{\mathrm{c}})$ taking into account the effect due to the crossing angle θ_{c} of the two beams, which do not collide exactly head-on. Run 2 had a peak instantaneous luminosity of $\mathcal{L} = 2{\cdot}10^{34}\ \mathrm{cm}^{-2}\ \mathrm{s}^{-1}$, and the same peak value has been reached in Run 3.

The number of collisions per unit time for a given physics process produced by the LHC, can be obtained simply multiplying the instantaneous luminosity times the cross section σ of the specific process:

$$\frac{dN}{dt} = \sigma \cdot \mathcal{L}. \tag{1.22}$$

Consequently, the total number of collisions is obtained integrating over time:

$$N = \sigma \cdot \int \mathcal{L} \, \mathrm{dt} = \sigma \cdot \mathscr{L}, \tag{1.23}$$

where $\mathscr{L}$ is called the integrated luminosity, usually expressed in inverse femtobarns or fb^{-1}, where 1 barn=10^{-24} cm^{-2}. Run 2 delivered $\mathscr{L}$=156 fb^{-1}, of which only for 140 fb^{-1} the condition of the detector were such that the data obtained can be used for physical analysis.
Within a bunch crossing multiple *pp* collisions can occur, and the number of these collisions is called in-time pile-up.
Also interactions coming from the previous or subsequent crossings can be mistakenly associated with the current event, and this is called out-of-time pile-up. Out-of-time pile-up has a significant impact because the electrical signals of parts of the detector may last more than the time that separates two bunches.
From now on we'll refer simply to the pile-up as the sum of the in-time and the out-of-time effects, and we can derive the average value for the pile-up μ with the following:

$$\langle \mu \rangle = \frac{dN}{dt} \times 25 \text{ ns}. \tag{1.24}$$

From equation 1.22, given the total *pp* inelastic cross section at $\sqrt{s}$= 13 TeV $\sigma = 78.1 \pm 2.9$ mb, as measured by the ATLAS collaboration [9], we derive an average $\langle \mu \rangle$ of about 40 for Run 2, with long tails up to 60 interactions per crossing.

1.3.2 High Luminosity LHC

During Run 2 the average pile-up per crossing has reached $\langle \mu \rangle = 36.1$ in 2018, and for Run 3 $\langle \mu \rangle \approx 55$ is expected. In the near future though the LHC is planned to be upgraded to the High Luminosity LHC (HL-LHC), where a series of changes for the beam pipe, like new superconducting magnets, and new technology for beam collimation, will allow an increase in the instantaneous luminosity up to a factor of five.
Indeed HL-LHC [10] is expected to reach a maximum instantaneous luminosity of $7.5 \cdot 10^{34}$ cm^{-2} s^{-1}, hence collecting an integrated luminosity of $\mathscr{L} = 3000\mathrm{fb}^{-1}$. According to plan, this luminosity we'll be reached in 2029, with the beginning of Run 4; we can see a detailed LHC timeline in figure 1.7, starting from Run 1 and going up to the HL-LHC Phase.
Also the average number of interactions per crossing will strongly increase, with $\langle \mu \rangle$ expected to reach values from 140 to 200.

This increase in the number of events produced will provide a much greater quantity of data for SM precision measurements and Beyond SM (BSM) searches, with peculiar interest for the Higgs couplings and self-coupling.
Focusing on the ATLAS experiment, the upgrade to HL-LHC poses a challenge for the Trigger and Data Acquisition System (TDAQ), since the increase in the amount of data present for each bunch crossing can't be handled by the system used at the present time, so that a series of upgrades is necessary. In the following of this thesis we'll explore this problem in more detail.

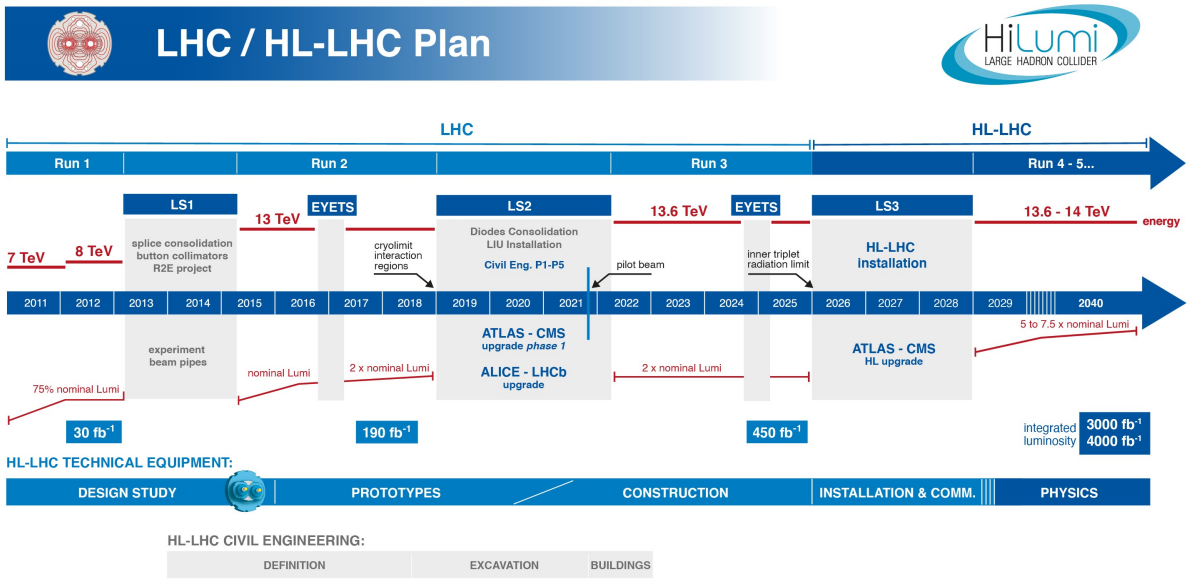


Figure 1.7: Timeline of the previous Runs and plan for the High Luminosity upgrade.

1.4 The ATLAS Detector

The ATLAS detector [11] is a cylindrical multi-purpose particle detector of 25 m in diameter, 44 m in length and weighting 7000 tons, located 100 m below ground.
We start by defining the coordinate system necessary to describe the ATLAS detector and the particles emerging from the $p-p$ collisions.

1.4.1 The ATLAS Coordinate System

First of all, the origin of the coordinate system is taken to coincide with the center of the detector, which is also defined as the nominal interaction point, since the physical interaction point for each collision will slightly deviate from the nominal one, because of the particular features of the beams. The z-axis lies on the axis of the detector, with the x-axis pointing towards the center of the LHC ring and the positive y-axis pointing upwards, as we can see in the illustration of figure 1.8.
We introduce spherical coordinates, with the azimuthal angle ϕ measured in the $x-y$ plane from the x-axis and the polar angle θ as the angle from the z-axis.
The polar angle θ is often replaced by a function of it, the pseudorapidity η, which is a high-energy approximation of the rapidity y:

$$y = \frac{1}{2}\ln\left(\frac{E+p_z}{E-p_z}\right) \quad \xrightarrow{\text{E>>m}} \quad \eta = -\ln\left(\tan\left(\frac{\theta}{2}\right)\right) \tag{1.25}$$

Angular distances are often quoted in terms of ΔR, which is defined as:

$$\Delta R = \sqrt{(\Delta\phi)^2 + (\Delta\eta)^2}. \tag{1.26}$$

The transverse momentum p_T of a particle is the projection of the momentum of a particle on the $x-y$ plane and the transverse energy is defined as $E_T = \sqrt{m^2 + p_T^2}$.

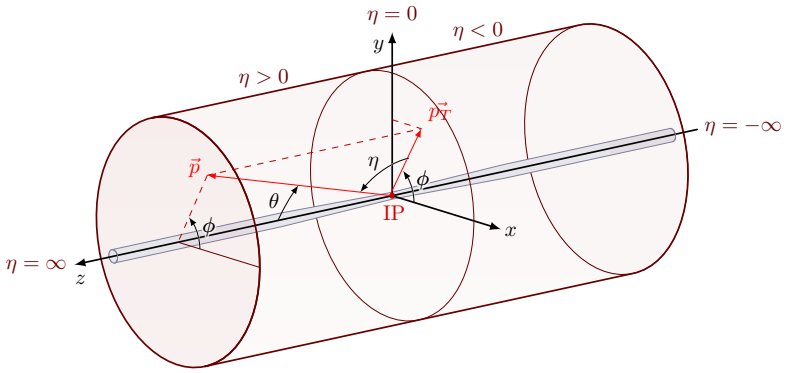


Figure 1.8: Illustration of the ATLAS coordinate system.

1.4.2 ATLAS Sub-detectors

The experiment is composed of several detector systems, disposed with cylindrical symmetry and each with a different task:

- we start with the **Inner Detector** (ID), situated directly around the beam pipe and enclosed by a thin superconductor solenoid magnet, which generates a uniform magnetic field of 2 T. This magnetic field bends the trajectory of the charged particles, producing approximately circular trajectories on the $x - y$ plane, and hence allowing a measurement of the particle's transverse momentum.

- The main role of the **Calorimeter System** is to absorb the particles and measures their energy, and also to reconstruct the electromagnetic and hadronic showers in order to perform particle identification. The calorimeters used in ATLAS are sampling calorimeters, which alternate layers of an absorber, a dense material used to degrade the energy of the incident particle, and layers of an active material medium that provides the detectable signal.

- Muons traverse the detector loosing only a small fraction of energy, so they are identified in the outermost detector layer, in the **Muon Spectrometer** (MS).
  The MS is used both for trigger and precision tracking purposes and is embedded in a toroid magnet system composed of a barrel toroid and two end-caps toroids, which generate inhomogeneous magnetic fields around 0.5 T and 1 T each.
  Both the barrel and the end-cap toroids are composed by 8 superconducting coils, disposed radially and symmetrically around the beam axis, causing the muons to curve in the planes passing through the z-axis, so that their p_T can be measured.

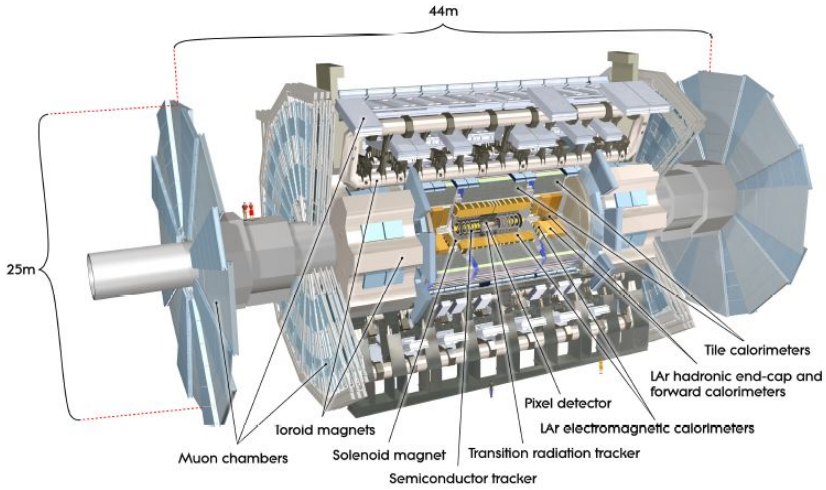


Figure 1.9: View of the ATLAS detector.

Each detector system is composed of a barrel part, coaxial to the beam line and enclosing the center of the detector, and two end-cap parts, disposed perpendicularly to the beam axis in order to cover regions with higher values of $|\eta|$.
We now go into more detail about the Inner Detector, since the work done in this thesis will mainly focus on the reconstruction of tracks, also called "tracking", from the information obtained in the ID.

1.4.3 Inner Detector

The ID is the innermost detector system, designed to provide precise tracking information for charged particles, in order to achieve high resolutions for the momentum and vertex reconstruction. The ID is divided into three sub-detectors, going from the inside to the outside we get: the Pixel detector, the Semi-Conductor Tracker (SCT) and the Transition Radiation Tracker (TRT), as it is represented in figure 1.10. We now give a brief overview about each one.

1. the **Pixel detector** consists of 4 cylindrical layers and 2 end-caps with 3 disk layers each, covered by Silicon Pixels and providing a high resolution in the reconstruction of the trajectories of charged particles.
   The pixels consist in reverse-biased pn-junctions, so that when a charged particle passes through, the material is ionized and electron-hole pairs are generated, so that this charge is then detected as an electric signal. The pixels that register a fraction of charge constitute a "hit", meaning that a particle crossed the detector nearby. Most particles will pass through multiple pixels, and the collection of pixels that present a fraction of the total charge generated is called "cluster".
   The layers in the barrel are located between a radius of 33.25mm and 122.5mm from the beam line and they ensure a coverage of $|\eta| < 2.5$. The closest layer, at only 3.3cm from the beam pipe, is the Insertable B-Layer (IBL), which was added in-between Run 1 and Run 2, and represents an important upgrade for the tracking system, since it plays a crucial role for b-tagging and for the vertex reconstruction.
   The IBL provides the highest granularity, with pixels of size 50 μm in the ϕ direction and 250 μm in the z direction, while the 3 remaining layers have pixels with dimensions of 50 μm $\times$ 400 μm, with radii of approximately 5 cm, 9 cm, 12 cm. The width of the pixels along the ϕ direction is also often called "pitch".
   This gives an expected hit resolution of 8μm and 10 μm in the ϕ-direction and 40μm and 115μm in the z-direction, for the IBL and the three other layers, respectively.
   The disks in the end-caps have radial extension 88.8mm$< r <$149.6mm and occupy 495mm$< |z| <$50mm.

2. The Semi-Conductor Tracker (**SCT**) is a silicon strip detector, formed by 4 layers in the barrel, with radii of 300 mm, 373 mm, 447 mm, 520 mm, and 2 end-caps with 9 disks. This detector layer covers up to $|\eta| <$2.5 and each silicon strip module is composed by two stereo strips, so that we get up to 8 measurements for each track

passing through. The strips are 12 cm long and have a pitch of 80 μm, so that in the barrel they provide a resolution of 17 μm in the ϕ direction and 580 μm along z.
In the end-caps the strips are disposed radially, and again we have resolutions of 17 μm in the ϕ direction and 580 μm in the radial direction.
Also for the SCT detector, multiple strips in both layers will be activated, forming again clusters of activated strips.

3. The Transition Radiation Tracker (**TRT**) is a gaseous detector that provides information about tracking and particle identification. It is composed of around 300k straw tubes disposed in parallel with the beam pipe for the barrel part, and around 50k placed radially for the end-cap parts.
The straws are filled with gas and in the centre of the tubes a metal wire is placed, which has a voltage of 1.5 kV with respect to the surface of the straw.
The straws are interleaved by a polypropylene film, in a way that when a particle crosses the boundary and the density of the material changes, a transition radiation of X-ray photons is emitted and detected.
The energy threshold above which a transition radiation is emitted, depends on the mass of the particle, for example electrons emit energies above 0.5 GeV, while charged pions do so only for energies above 130 GeV: transition radiation thus provides a method to distinguish electrons from pions over a wide range of energies
The TRT covers an angular region of $|\eta|$ <2 and gives on average 30 two-dimensional space points with 130 μm resolution.

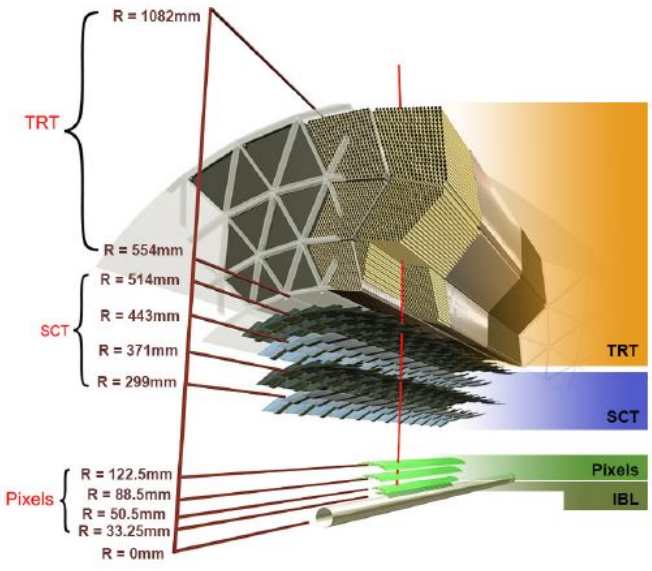


Figure 1.10: Overview of the Inner Detector.

1.5 The ATLAS Trigger System

In 2018, during Run 2, the average pile-up observed was $\langle\mu\rangle = 36.1$; considering that a collision between the bunches of protons happens every 25 ns, this amounts to 1.5 billions proton-proton interactions per second. If we consider the data volume registered by the ATLAS detector, this rate of interactions translates to about 60 TB per second, which of course cannot be handled.
For this reason the Trigger and Data Acquisition (TDAQ) system is a crucial component of the ATLAS experiment, since it decides whether or not to record the data from a given collision, with the goal to reject the uninteresting events while preserving as much as possible the signals coming from relevant physical processes. The TDAQ is also responsible for the real-time reconstruction of the data produced by the collisions, which is used to take the trigger decision; for the events selected by the HLT, a more detailed reconstruction will be done for the offline analysis.
In figure 1.11 we see a schematization of the trigger system, which is composed of two stages:

1. the Level-1 (L1) trigger,
2. the High Level Trigger (HLT).

1.5.1 Level-1 Trigger

The L1 trigger [12] is a hardware system that operates a selection of the events, based on the reduced-granularity information coming from the calorimeters and the muon detectors. The L1 trigger accepts events up to a maximum rate of 100 kHz, strongly reducing the bunch crossing rate of 40 MHz, with a latency of 2.5 ms.
The decision at Level-1 is taken utilizing information coming from three different categories of triggers:

- L1 calorimeter (L1Calo) trigger
- L1 muon (L1Muon) trigger
- L1 topological (L1Topo) trigger

The L1Calo trigger takes the signal coming from the calorimeters and identifies candidates for electrons, photons, τ-leptons and hadronic jets above programmable thresholds of transverse energy. The L1Muon trigger uses hits from the Muon Spectrometer, reconstructing the tracks and computing the resulting momentum of the muons. The L1Topo trigger operates a selection based on topological requirements, looking for geometric or kinematic combinations between the objects provided by the L1Calo and the L1Muon.
The Central Trigger Processor (CTP) receives inputs from the L1Calo, L1Muon, L1Topo and takes the L1 trigger decision. The CTP is also responsible for applying ”dead time”, which is a mechanism to limit the number of L1 accepts and avoid overlapping read-out

windows. While the L1 is taking the trigger decision, the data is stored on the on-board memory of the detectors; if the event is selected with an L1 accept, the data from the entire detector is then sent to read-out buffers (ROBs), ready to be passed on to the next level in case of an HLT accept.
L1 also selects some regions of interest (RoIs), as intervals of η and ϕ of the detector, where there's a higher probability to find relevant physical processes, so that it may be sufficient to investigate these regions instead of processing all the data coming from the detector. The RoIs reduce the amount of dadeadta to the 5% of the total data volume, and these regions of the detector are passed to the HLT.

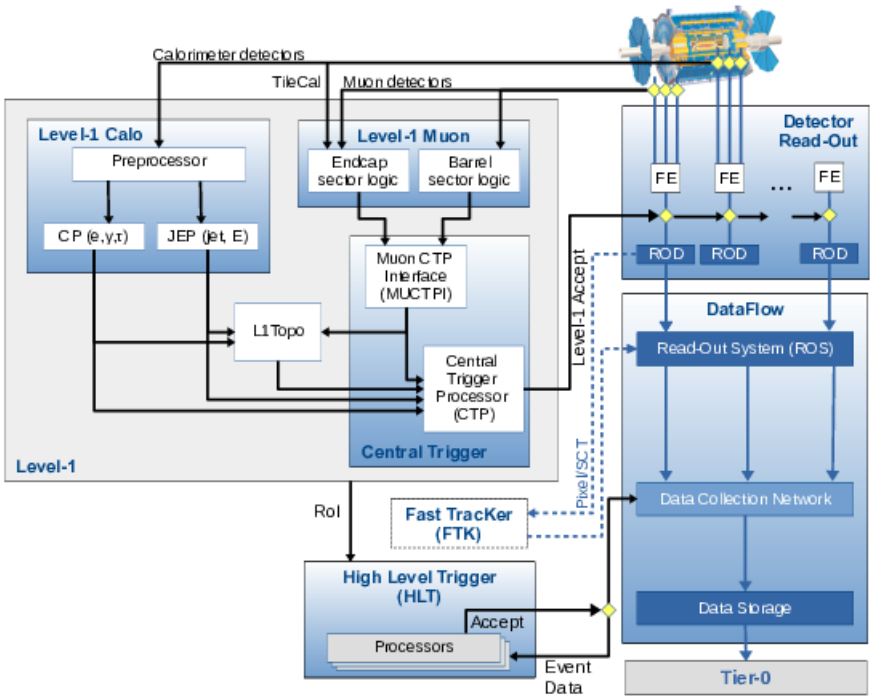


Figure 1.11: Schematic view of the ATLAS trigger system.

1.5.2 High Level Trigger

The HLT is a software trigger level, with a dedicated dedicated CPU farm used to run fast trigger algorithms for early rejection, and then more precise algorithms, similar to those used for offline reconstruction.
The HLT receives as input, data coming from events which have been already selected at L1, and then applies another selection, so that the event rate can be further reduced, thus obtaining an average output of 1.2 kHz and a latency in the order of ~1 s.

This is the first level where the information from the ID is available; that's because the ID generates a huge amount of data, which cannot be handled for all the events so that the pre-selection of the L1 is necessary.
The HLT processes data coming either from RoIs selected at L1, or from the full detector, and if the trigger decision is to accept the event, the data is then sent to the Tier-0 computing center at CERN, where the offline analysis of the events starts.
Within the HLT progressively more complex algorithms are deployed to reduce the rate of the events, so that the fastest algorithms are executed first, allowing the more complicated and time consuming algorithms to operate on a reduced rate of events.
The HLT processes multiple items, targeting photons, electrons, muons, taus and jets that may have come from an interesting physical process.

- The photon identification algorithm relies on cuts of calorimetric variables, defined using the energy deposit in the cells of the Electromagnetic Calorimeter.
- Electrons are selected at the HLT using a likelihood-based criteria which takes as input the electromagnetic shower shape and the tracking information.
- Muons are reconstructed by combining the inner detector and muon spectrometer tracks.
- Tau candidates are identified using an algorithm that takes calorimeter and track quantities as inputs.
- Jets are found from the reconstructed tracks of the event, their selection is based on their energy and whether the jet is b-tagged or not.

In this thesis I want to focus mainly on the real-time reconstruction of tracks within the HLT, and also on the triggers targeting b-jets, which utilize the reconstructed tracks.
Figure 1.12 shows the workflow adopted for the reconstruction of tracks at the HLT level in Run 2, which is mainly divided in two parts, the Fast Track Finder (FTF), where tracks are obtained using the points coming from ID data, and the Precision Tracking stage, where a further selection is applied to the FTF track candidates, choosing the higher quality ones. These tracks are the final result of the track reconstruction within the HLT and are used for the trigger selections.
From Run 2 to Run 3 there has been a change in the strategy adopted for the HLT tracking, with a new stage added at the beginning of the process, where a b-tagging algorithm is used on FTF tracks and a pre-selection of the events is applied. In this way the event rate is reduced, allowing a more precise and CPU-intense reconstruction in the following stage.
Figure 1.13 shows a comparison between the processing chains used in Run 2 [13] and Run 3 [14], where the added steps are depicted in yellow.
In the following chapter I'll explain the impact of the pre-selection added in Run 3 on the acceptance of $HH \to 4b$ events, and in chapter 3 I'll cover in more detail each step of the workflows shown in figure 1.12 and figure 1.13.

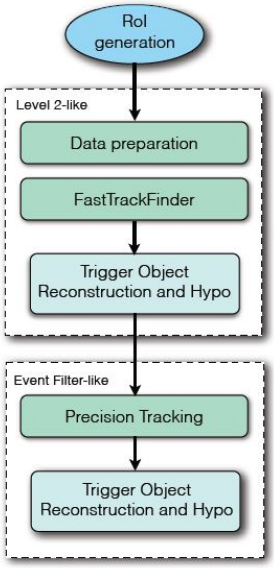


Figure 1.12: Workflow of the HLT used during Run 2.

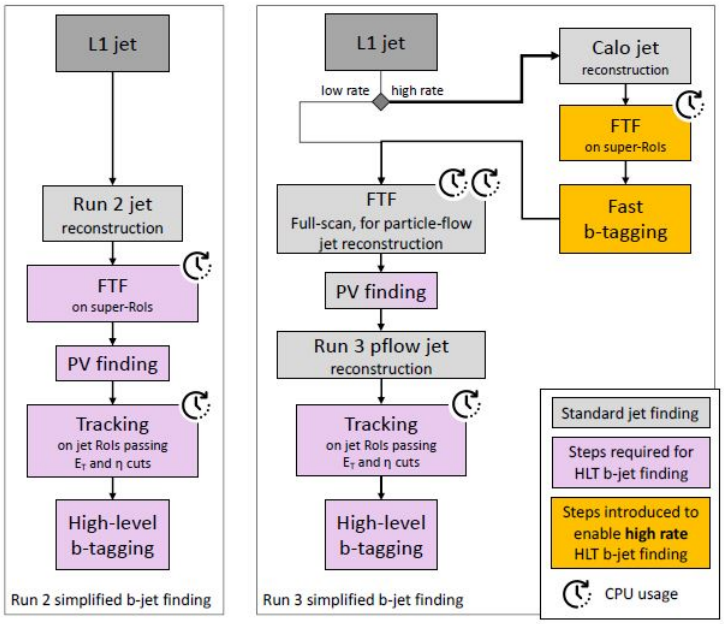


Figure 1.13: Comparison between the HLT workflows in Run 2 and in Run 3.

Chapter 2

Higgs pair production and decay to $b\bar{b}b\bar{b}$.

In this chapter I talk about the production of a pair of Higgs bosons, and then focus mainly on the consequent decay of each H in a pair of b-quark and $\bar{b}$-quark. First of all, I present the non-resonant production of a Higgs pair, which can be generated from a single H thanks to the trilinear self-coupling term present in the Higgs potential. This interaction though is extremely rare and no Higgs pair has been observed yet at LHC, but this channel represents one of the goals of the collaboration.
I then discuss the leading non-resonant production modes of HH, which are the gluon-gluon fusion (ggF) and the vector boson fusion (VBF), as well as the hypothetical resonant production, with a Higgs pair being produced from a heavy beyond SM spin-0 particle.
The main searches for the observation of a Higgs pair are $b\bar{b}\gamma\gamma$, $b\bar{b}\tau^+\tau^-$ and $b\bar{b}b\bar{b}$, the third being the one with the highest branching ratio. It is possible to combine the results for each analysis, hence obtaining a lower limit for the signal strength μ and a narrower interval for the self-coupling modifier κ_λ. In the following I present the results of the combination of the searches on Run 2 data.
The increased luminosity and center-of-mass-energy that will be achieved at the HL-LHC will cause a way larger number of events produced, so that the observation of H pairs may be possible. I show a study done on Run 2 data, scaled to resemble the data obtained from HL-LHC, producing prospects on the signal significance for HH production; the results of the analysis show optimistic results for the discovery of Higgs pairs at HL-LHC.
I then focus on the $HH \to b\bar{b}b\bar{b}$ process, and describe the importance of the trigger system for this channel. The great majority of the events with multiple b-jets in the final state are due to QCD processes, and so it is necessary to come up with a trigger strategy that could reject the QCD background, while being compatible with the CPU and latency requirements of the system. In this context I present my study on the impact of lower-momentum thresholds for the jet selection within the HLT, and look at how the acceptance of $HH \to 4b$ changes when the p_T values are lowered. Finally I present the effect of the changed HLT strategy in Run 3, introduced in the previous chapter, on the acceptance of $HH \to 4b$ events.

2.1 HH production

Since the discovery of the Higgs boson at LHC in 2012 [15], the properties of this new boson have been extensively tested, and thus far, all the measurements of its quantum numbers and its couplings are in agreement with the SM, so that no signs of physics beyond the Standard Model (BSM) have been yet observed.
One of the main goals for the future of the ATLAS and CMS collaborations is to measure the self-coupling of the Higgs boson, which is related to the trilinear self coupling λ_{HHH}, and contributes to production of boson pairs (HH). The self coupling coefficient λ_{HHH}, is of particular interest because directly connected to the shape of the Higgs scalar field potential, so that its measurement would provide an important test of the electroweak symmetry breaking mechanism.
We now look in more detail at the non-resonant production of Higgs pairs within the SM, and later analyze the possibility of resonant production from BSM particles.

2.1.1 Non-resonant HH production

We now look at some consequences of the introduction of the Higgs boson, described in-chapter 1, in order to confirm the importance of the study of HH non-resonant production. Indeed the Higgs mechanism [16] assumes the existence of a complex doublet of fields Φ, which is subject to the following potential:

$$V(\Phi) = -\mu^2\Phi^\dagger\Phi + \lambda(\Phi^\dagger\Phi)^2 \tag{2.1}$$

with $\mu^2, \lambda > 0$. The potential is minimal for non-zero field values such that:

$$|\Phi|^2 = \frac{\mu^2}{2\lambda} = \frac{v^2}{2} \tag{2.2}$$

where v is called the vacuum expectation value or VEV. When the symmetry of the potential V is spontaneously broken to a ground state, the degrees of freedom of the field Φ give rise to the longitudinal polarizations of the W and Z bosons, which gain mass, and to a scalar field H corresponding to the Higgs field. Expanding Φ near the minimum of the potential, and choosing the unitary gauge, we have:

$$\Phi(x) = \frac{1}{\sqrt{2}}\begin{pmatrix} 0 \\ v + H(x) \end{pmatrix} \tag{2.3}$$

If we then compute the potential V using this expression for $\Phi(x)$, we obtain that the SM Lagrangian will contain the following terms:

$$\mathcal{L} \supset \lambda v^2 H^2 + \lambda v H^3 + \frac{\lambda}{4}H^4 \tag{2.4}$$

so that the first term (H^2) gives rise to the Higgs mass $m_H = \sqrt{2\lambda v^2}$, and the trilinear ($H^3$) and quadrilinear ($H^4$) self-interactions are governed by the value of λ, and thus they are directly connected to the shape of the potential in equation 2.1.

We can now understand the relevance of the experimental study of HH production, because the observation of this process would enable the measurement of λ, for which the knowledge of both the mass and the trilinear coupling is necessary, hence providing a test of the consistency of the SM. The study of the Higgs boson self-coupling is also relevant in the context of cosmology, since it may play a role in the inflation mechanism, and in the baryogenesis of the primordial universe.
We now take a look at the channels that mainly contribute to the production of HH: within the SM the gluon-gluon fusion process (ggF) is responsible for more than the 90% of the production of HH from pp collisions, and at leading order (LO) the Feynman diagrams that describe the process are shown in figure 2.1.

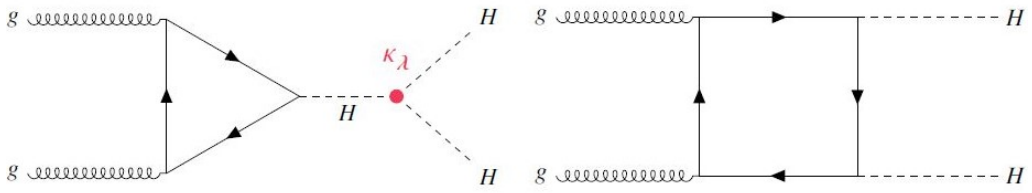


Figure 2.1: Feynman diagrams for the ggF production of a pair of Higgs bosons.

On the left of figure 2.1 we have a triangle diagram, and then the contribution of the trilinear self-coupling λ_{HHH}, since we have the production of HH starting from a single boson. On the right we have instead a box diagram, with a loop of 4 fermions and the production of each H due to the Yukawa coupling. The main contribution for both loops is given by the top quark, since it's the fermion with the highest mass in the SM, and hence the fermion with higher Yukawa coupling with the Higgs.
These two configurations interfere destructively, hence explaining the very low cross section for the HH production. The predicted SM cross-section for the ggF production is $\sigma_{ggF}^{SM} = 31.05 \pm 3.0\%$ fb at next-to-next-to-leading order (NNLO) [17], considering the Higgs boson mass $m_H = 125$ GeV and $\sqrt{s} = 13$ TeV.
The next leading production mode for a Higgs pair is the Vector Boson Fusion (VBF), which contributes for 5% of the HH production rate within the SM.

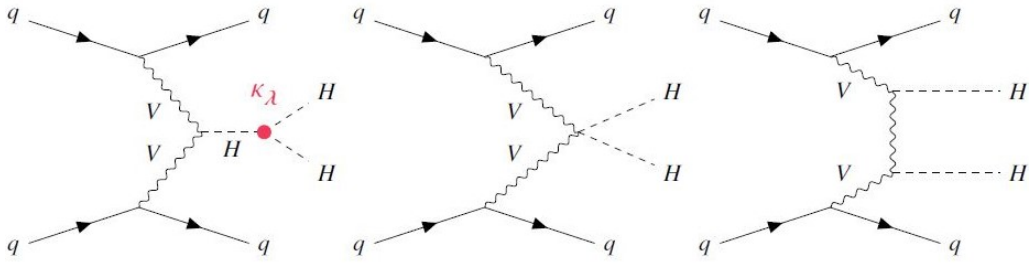


Figure 2.2: Feynman diagrams for the VBF production of a pair of Higgs bosons.

In figure 2.2 we see the leading order Feynman diagrams for the non-resonant production of HH, with the first one on the left showing a trilinear self-interaction of the Higgs, and the remaining two involving the couplings between the Higgs and vector bosons, such as Z and $W^{\pm}$.
The SM cross-section for the VBF production, predicted at next-to-next-to-next-to-leading order (N3LO) [18], again with $m_H = 125$ GeV and $\sqrt{s} = 13$ TeV, is $\sigma^{SM}_{VBF} = 1.73 \pm 2.1\%$ fb.
As of today, the pair production of Higgs bosons hasn't been observed yet at LHC, as its cross-section is 3 orders of magnitude smaller than the single Higgs production, but from the analysis of the available data, it is possible to put constraints on:

$$\kappa_\lambda = \frac{\lambda_{HHH}}{\lambda^{SM}_{HHH}} \quad \text{and} \quad \mu = \frac{\sigma_{HH}}{\sigma^{SM}_{HH}}, \tag{2.5}$$

where at the numerator we have the quantities measured from the data and at the denominator the theoretical values obtained from the SM, and we refer to κ_λ as the self-coupling modifier, while μ is the signal strength of the HH production.
We'll later observe the results obtained for κ_λ and μ from the analysis of Run 2 data.

2.1.2 Resonant HH production

A pair of Higgs bosons may also be produced with the decay of hypothetical heavy resonances [19], and many BSM theories predict the existence of such heavy particles.
The possible resonances considered are divided in two categories:

- Spin-0 bosons, that we'll refer to as X, are predicted by two-Higgs-doublet models such as the Minimal Supersymmetric Standard Model (MSSM).
- Spin-2 Kaluza-Klein graviton G^*_{KK}, predicted by the Randall-Sundrum (RS) model.

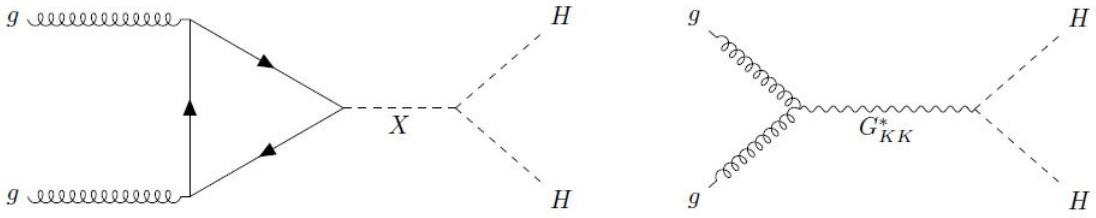


Figure 2.3: Feynman diagrams for the resonant production of a pair of Higgs bosons.

In both cases, the results that I'll report in this chapter are obtained only considering the gluon-gluon fusion production mode, as shown in figure 2.3.
All the searches carried out by the ATLAS and CMS collaborations have found results consistent with the SM prediction that no heavy resonances exist, and from the data it has been possible to put constraints on the mass of these hypothetical particles.

2.1.3 *HH* searches

We now go through the decay modes that are mainly used for the HH searches [20], looking at the branching ratios of the processes and the most relevant backgrounds for each search. The results that will be presented in the following sections are obtained from the analysis of the full Run 2 dataset of 139 fb^{-1}, so we also explain how the events will be selected from the data for each decay mode.

$b\bar{b}\gamma\gamma$: even if the $HH \to b\bar{b}\gamma\gamma$ decay mode is very rare, with a branching ratio of only 0.26%, the experimentally clean signature characterized by the presence of the 2 photons, makes it one of the most sensible channel for the HH search.
The selection requires the presence of two photons and two b-tagged jets, each compatible with the decay of the Higgs boson. The main background processes are the production of photon pairs and jets, and events with the decay of a single Higgs to a pair of photons. For the resonant search, also the non-resonant production is considered background.

$b\bar{b}\tau^+\tau^-$: the $HH \to b\bar{b}\tau^+\tau^-$ decay mode has a branching ratio of 7.3%, with the $b\bar{b}$ decay increasing the total branching ratio while the $\tau^+\tau^-$ decay enables to distinguish the process from the QCD background. The search for this channel targets both fully-hadronic and semi-leptonic final states with two τ particles, which we indicate with $\tau_{\text{had}}\tau_{\text{had}}$ and $\tau_{\text{lep}}\tau_{\text{had}}$. In the $\tau_{\text{had}}\tau_{\text{had}}$ situation both τ decay hadronically, generating two τ-jets, while for $\tau_{\text{lep}}\tau_{\text{had}}$ one τ generates a jet and the other one decays leptonically, producing a lepton and neutrinos.
For both channels also two b-tagged jets are required for the selection.
The main background for this search come from $t\bar{t}$ jets and Z + heavy flavour production.

$b\bar{b}b\bar{b}$: while this decay mode has the highest branching ratio of 33.9%, $b\bar{b}b\bar{b}$ also has the largest SM background, due to the large production of QCD multi-jet processes. So an efficient identification of b-jets at the trigger level is required, as well as dedicated strategies to separate the background from the HH signal.
The current searches cover two different topologies, the 'resolved' topology, where all four b-jets can be separately reconstructed, and the 'boosted' topology, where the H decays with a high momentum, resulting in collimated decay products, so that the HH system is reconstructed as two large-radius jets.

2.2 Results from HH data

We now present the results of the combination of the searches for the Higgs pair production [21], obtained using the 139 fb^{-1} of data from Run 2 and assuming the mass of the Higgs $m_H = 125$ GeV.
Later we also show the prospects for the HH search at HL-LHC, for which it is expected to obtain 3000 fb^{-1} of data, hence strongly increasing the probability of observing the Higgs pair production.

2.2.1 Results from Run 2 data

We start with the analysis of the non-resonant production, which uses the results obtained from both the $b\bar{b}\gamma\gamma$ and $b\bar{b}\tau^+\tau^-$ channels. A statistical combination of the two searches allows to set upper limits for the signal strength and the cross-section for the non-resonant ggF and VBF production. The same analysis conducted on Run 2 events is also carried out for Monte Carlo simulations of the events, so that we can double check the results, and confront them with the theoretical prediction.
The MC samples are generated with software developed within ATLAS, with programs that deal with specific tasks, like the generation of the event, in this case the non-resonant ggF and VBF productions of HH, the parton showering and hadronization processes, and then the simulated interaction between the particles and the detector.
In order to obtain realistic results from the Monte Carlo samples we also need to generate the background events, which, depending on the search, are modeled using simulations or are estimated with data-driven techniques.

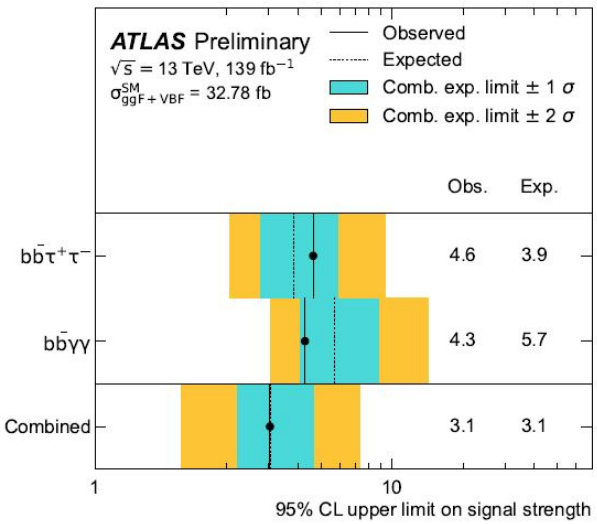


Figure 2.4: Upper limit for the signal strength μ, at a 95% confidence level, for the single searches $b\bar{b}\gamma\gamma$ and $b\bar{b}\tau^+\tau^-$, and their statistical combination.

In the following we refer to the observed results as the measurements obtained from the analysis of the Run 2 data, while the expected results are obtained from the MC simulated samples.
The observed (expected) result for the combined upper limit of the HH signal strength μ, at a 95% confidence level, is $\mu = 3.1(3.1)$. In figure 2.4 we see the upper limits obtained separately from the $b\bar{b}\gamma\gamma$ and $b\bar{b}\tau^+\tau^-$ channels, and then their combination.
The observed (expected) combined 95% confidence level upper limit for the cross-section is found to be 92.0 fb (92.2 fb), when the theoretical prediction is $\sigma^{\text{SM}}_{\text{ggF+VBF}}(HH) = 32.78$ fb.
Constraints can also be found for the Higgs boson self-coupling modifier κ_λ; Monte Carlo samples of HH production are generated with $\kappa_\lambda = 1$ and 10, for the ggF production , and using $\kappa_\lambda = 0, 1, 2$ and 10 for the VBF production.
A re-weighting method is then used, which consists in performing a linear combination of the samples generated with different values of self-coupling modifier in order to obtain scale factors as a function of κ_λ.
It's now possible to obtain the upper limit at 95% confidence level for the HH cross-section $\sigma_{\text{ggF+VBF}}(HH)$ as a function of κ_λ. The limits obtained for the $b\bar{b}\gamma\gamma$ and $b\bar{b}\tau^+\tau^-$ searches are then combined, as shown in figure 2.5, where the theoretical prediction for the cross-section as a function of κ_λ is overlaid.
The points of intersection between the theoretical prediction and the observed combined upper limits determine the allowed range for κ_λ, which are found to be $-1.0 \leq \kappa_\lambda \leq 6.6$ $(-1.2 \leq \kappa_\lambda \leq 7.2)$.

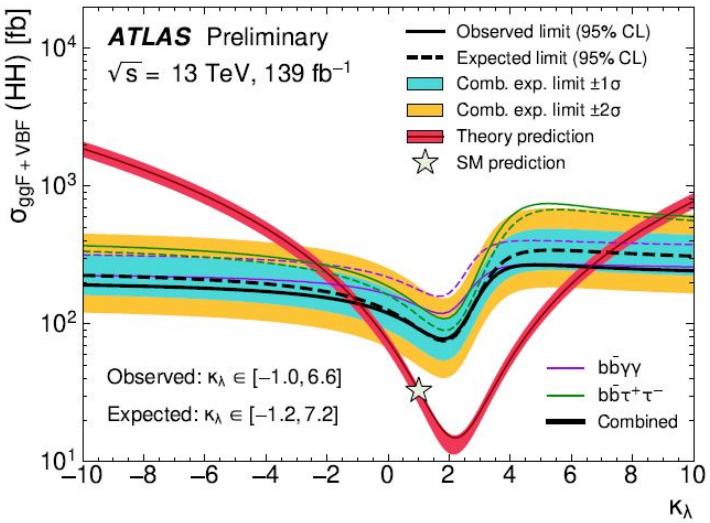


Figure 2.5: The 95% CL upper limits for the ggF and VBF HH cross-section are shown. Observed and expected upper limits are portrayed, as well as the theoretical prediction for $\sigma_{\text{ggF+VBF}}(HH)$ as a function of κ_λ

We now show the results of the combination of the searches for the HH resonant production, which utilize data from $b\bar{b}\gamma\gamma$, $b\bar{b}\tau^+\tau^-$ and also $b\bar{b}b\bar{b}$. With resonant production we refer to the production of a heavy spin-0 boson X as a result of a pp collision, which then decays in a Higgs boson pair, $X \to HH$, which consequently decay in the before mentioned channels. Only the ggF production for X is considered.
Also in this case MC samples for both the signal event and the background are produced; multiple Monte Carlo samples are generated, with m_X varying between 251 GeV and 3 TeV, setting X's width to 10 MeV.
As we can see in figure 2.6, the observed results, found from the combination of the three searches, are compatible with the SM within 2σ, with the largest deviation with respect to the predicted result found for $m_X = 1.1$ TeV.

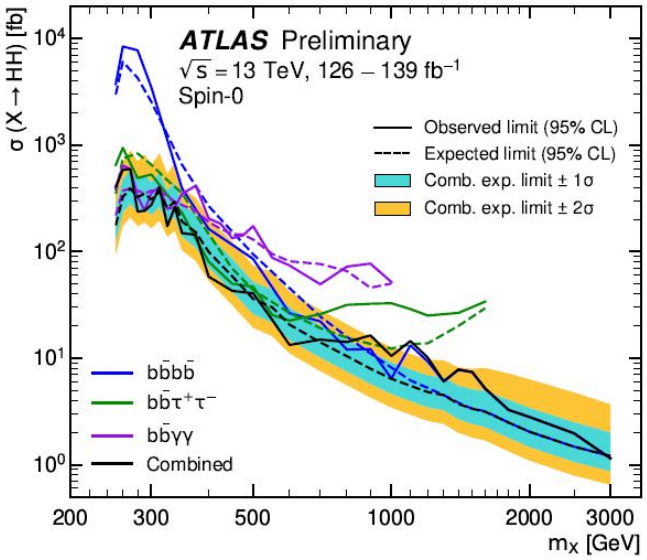


Figure 2.6: 95% CL upper limits for the cross-section of $X \to HH$ for each search and then their statistical combination.

2.2.2 Prospects for HL-LHC

The HL-LHC project is expected to start in 2029, with the plan of being operational for 10 years, delivering a total integrated luminosity of 3000 fb^{-1} at a center-of-mass energy of $\sqrt{s} = 14$ TeV. Higher energies result in an increase of $\sigma_{\text{ggF+VBF}}(HH)$, which, combined with the strong increase of the luminosity, will cause a much higher probability for a Higgs pair production with respect to the possibility at LHC.
In the following we report a prospect for the results that may be obtained at HL-LHC [22], based on a study extrapolated from the Run 2 data, which has been re-scaled in order to match the luminosity and energy settings of HL-LHC. Indeed all Run 2 analysis

distributions are scaled by a multiplicative factor defined as the ratio between the target HL-LHC integrated luminosity of 3000 fb^{-1} and the Run 2 integrated luminosity of 126 fb^{-1}, and also an additional scale factor to account the increase from $\sqrt{s}$ = 13 TeV to $\sqrt{s}$ = 14 TeV.
Since it's not known how the systematic uncertainties will change from Run 2 to HL-LHC, multiple scenarios are considered, which include that the systematic will remain unchanged, they will be halved, or they will be scaled down, due to the larger dataset available. Also the situation with no systematic uncertainty is considered.
In figure 2.7 the prospects for the upper limits of the signal strength μ at HL-LHC are shown; the results are obtained from the combination of the three searches analyzed and the predictions for each uncertainty are shown.

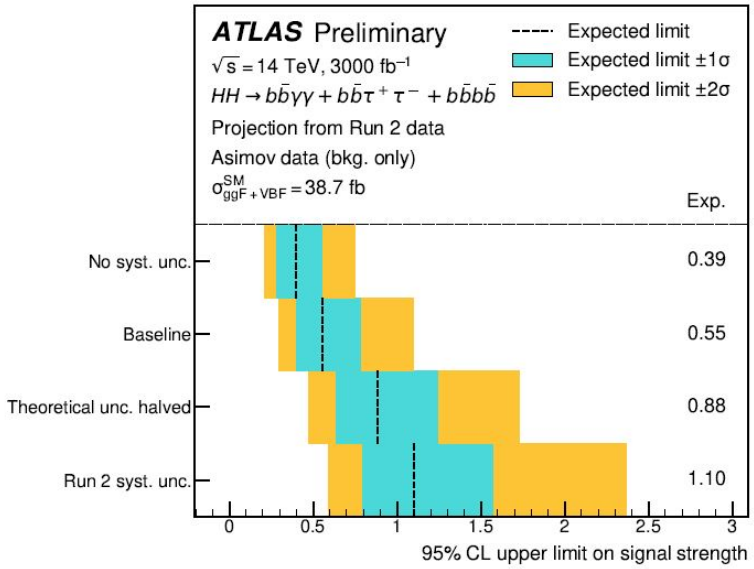


Figure 2.7: Upper limit for the signal strength μ, at a 95% confidence level, for the single searches $b\bar{b}\gamma\gamma$ and $b\bar{b}\tau^+\tau^-$, and their statistical combination.

We now present the prospects for the statistical combination of the $b\bar{b}\gamma\gamma$, $b\bar{b}\tau^+\tau^-$ and $b\bar{b}b\bar{b}$ channels, with the expected HH discovery significance shown in table 2.1, computed for each hypothesis on the systematic uncertainty.
So in the optimistic estimation of the uncertainties it is prospected that evidence for the production of HH will be obtained at HL-HLC, since the signal significance is greater than 3σ, but in order to obtain the proof of the observation of an event, a signal significance of at least 5σ is required. In the next years anyway, further improvements of the offline reconstruction will be added, making steps towards the discovery of non-resonant HH production at the HL-LHC.

Uncertainty Hypothesis	$b\bar{b}\gamma\gamma$	$b\bar{b}\tau^+\tau^-$	$b\bar{b}b\bar{b}$	Combination
No Systematic Unc.	2.3	4.0	1.8	4.9
Scaled Down Unc.	2.2	2.8	0.99	3.4
Halved Unc.	1.1	1.7	0.65	2.1
Run 2 Unc.	1.1	1.5	0.65	1.9

Table 2.1: Signal significance in units of σ for each uncertainty assumption. Results for each search are shown as well as their combination.

2.3 Impact of the trigger on $HH \to 4b$

The detection of events with multiple b-jets, specifically $HH \to 4b$, represents a challenge for the ATLAS trigger system, because these events require the reconstruction of multiple b-tagged jets at low momenta, in a phase-space region which is dominated by QCD interactions. It becomes then necessary to carefully elaborate a trigger strategy to enhance the possibility of distinguishing the $HH \to 4b$ signal from the overwhelming QCD background, in order to obtain a compromise between the acceptance of the physical signal and a workload that can be managed by the CPU resources of the HLT.
For example at the HL-LHC, the increased luminosity will also cause events with a way higher density of tracks, so that the fast tracking algorithms will consequently require more time to complete the track reconstruction; we'll study in more detail the relation between the density of tracks, or more precisely the average pile-up $\langle\mu\rangle$, with the timing of the tracking algorithms in chapter 3. In this situation is then necessary to reduce the workload of the CPU farm, either by looking for more efficient and fast algorithms, or also by reducing the rate of events processed at the trigger level. A possible way to reduce the rate is to raise the minimum p_T thresholds to accept an event; this approach has been tested on the Run 2 dataset scaled to $\sqrt{s} = 14$ TeV and integrated luminosity of 3000 fb^{-1} [23].
This study is based on a selection of events with four b-tagged jets, where two pairs of them have an invariant mass consistent with the one of the Higgs boson. In figure 2.8 we have the events that passed the four b-jet trigger, and we can see the main sources of background, with QCD multi-jets contributing for the 90% and $t\bar{t}$ jets for the remaining 10%.
We now look at the effect that raising the p_T thresholds at trigger level can have on the sensitivity of the $HH \to 4b$ measurement. In figure 2.9 we see how increasing the p_T thresholds for the trigger level selection negatively impacts the sensitivity of the $HH \to 4b$ search, since the 95% CL upper limits for the signal strength of $HH \to 4b$ increase as a function of the thresholds on the momenta.
Also the sensitivity for the Higgs self-coupling λ_{HHH} shows a degradation for increased value of the p_T thresholds, as we can see in figure 2.10.
We conclude that while lowering the thresholds on the momenta may reduce the trigger rates to an acceptable level, we have to discard this option since it impact the acceptance of the signal.

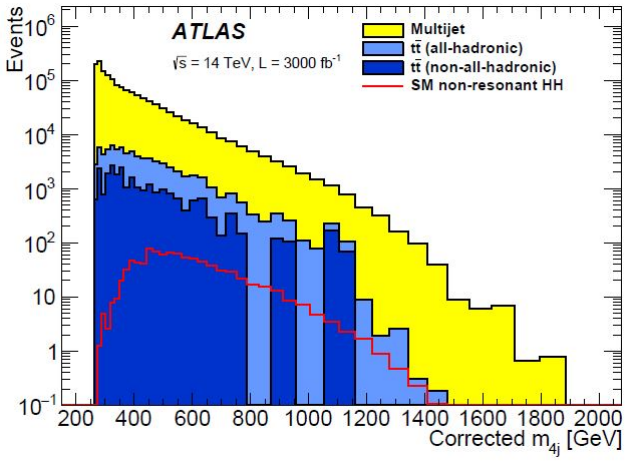


Figure 2.8: Events that passed the four b-jet trigger for the HL-LHC-like dataset. The background for the $HH \to 4b$ signal consists in QCD multijets and $t\bar{t}$ jets.

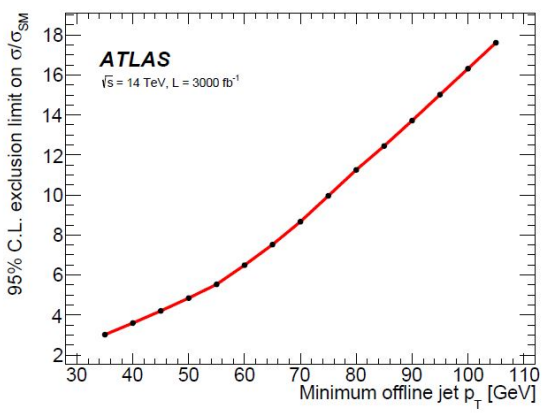


Figure 2.9: Upper limits for the $HH \to 4b$ signal strength as a function of the p_T threshold.

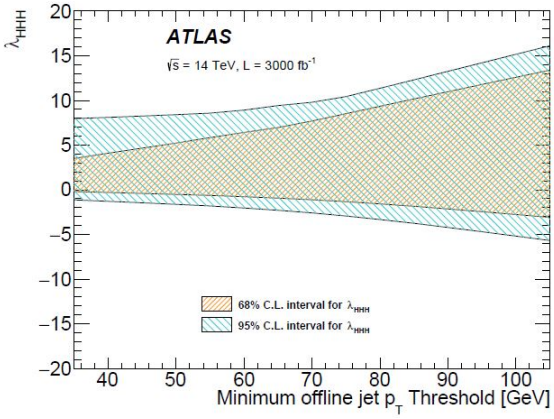


Figure 2.10: 95% and 68% CL intervals for λ_{HHH} as a function of the p_T threshold.

The results shown in figures 2.9 and 2.10 are obtained from an offline analysis, but the cut operated on the jets according to their p_T simulates the action of the trigger, and so we can interpret these results in the context of the HLT.
In the following I show a study conducted on a sample of MC simulated $HH \to 4b$ events, looking instead at the effect that lowering the momentum thresholds can have on the acceptance of $HH \to 4b$, in the context of the Run 2 trigger selection.
Later I also present the impact of the new trigger strategy adopted in Run 3 on the detection of $HH \to 4b$.

2.3.1 $HH \to 4b$ acceptance with the Run 2 trigger

We have just seen that raising the p_T thresholds at trigger level has negative impact on the $HH \to 4b$ acceptance, but on the other hand it also causes a decrease of the trigger rate, hence reducing the CPU workload. If we instead try to reduce the p_T thresholds, we expect an opposite effect: an increase of the signal acceptance while the CPU workload will increase, due to an increased trigger rate.
If a way was found to save some processing time within the HLT, then with the spare CPU resources it would be possible to lower the momentum threshold and in this way extend the physical reach at trigger level. We then have to ask ourselves whether is it worth it to make use of these computing resources to lower the p_T requirements; in this work only a partial answer to this question will be given, because the study that I'll show is only based on the acceptance of the $HH \to 4b$ signal, while the determining parameter would be the ratio between this acceptance and the amount of background events accepted by the trigger. Indeed lowering the thresholds we are also selecting a bigger amount of noise events, which may end up worsening the situation.
After this premise I am going to study how the acceptance of $HH \to 4b$ changes when lowering the thresholds; to do so I artificially apply the trigger requirement on a Monte Carlo sample of $HH \to 4b$ events and take a look at how many events survive the selection. First of all we need to decide the trigger selections at the HLT level that suit the search of the decay of the Higgs pair into b-quarks; the selection we are looking for has to require multiple jets with low momenta, which, if possible, should also be b-tagged, in order to have a higher probability of picking $HH \to 4b$ events. From the list of the trigger selections from the HLT of Run 2, the ones which come closer to our requirements are the following:

- selection of the events with at least one b-tagged jet with $p_T > 175$ GeV and at least one b-tagged jet with $p_T > 60$ GeV.
- selection of the events with at least two jet with $p_T > 250$ GeV and at least one jet with $p_T > 125$ GeV.

For each event in the Monte Carlo sample I have access to the information of the momentum of each reconstructed jet, so that I can count how many of them have transverse momentum satisfying the thresholds.
Regarding the flavour of the jet, in the sample I can also find the probabilities p_b, p_c and p_u obtained with the b-tagging algorithm DL1d, so that I can construct the b-tagging discriminant D_b:

$$D_b = \ln\left(\frac{p_b}{(1-f_c)\,p_u + f_c\,p_c}\right), \tag{2.6}$$

where the value of f_c is found looking for the optimal balance between the importance of the c and light-flavour jets rejection for the DL1d algorithm, and is set to $f_c = 0.018$.

It's now necessary to compute the value of $D_{HH\to 4b}$ relative to a chosen efficiency of the b-tagging on the $HH \to 4b$ sample, so that between the jets with $D_b > D_{HH\to 4b}$ a chosen percentage of them actually belongs to a b-jet.
This value for the efficiency is also called Working Point (WP), and in this case we choose the 60% WP corresponding to:

$$D_{HH\to 4b} = 1.92. \tag{2.7}$$

Now we can enact the b-tagging on the sample, classifying every jet with $D_b > D_{HH\to 4b}$ as a b-jet, knowing that we have an integrated 60% efficiency in doing so.
I now proceed with computing the performance of the two triggers for lowered momentum thresholds; to do so I iteratively reduce the thresholds of 5% and continue with counting the number of jets satisfying the requirements for each event. In figure 2.11 we have the results for the first selection, showing the dependence of the acceptance of the trigger, computed as the ratio between the events that are accepted and the total number of events, as a function of the reduced threshold. As we expected, we see a noticeable increase of the acceptance with the reduction of the p_T requirements, with only 7.5% of the $HH \to 4b$ events surviving the first trigger with (175 GeV, 60 GeV) momenta thresholds, while more than 50% makes the cut with (87.5 GeV, 30 GeV) momenta. In figure 2.12 I show the analogous results

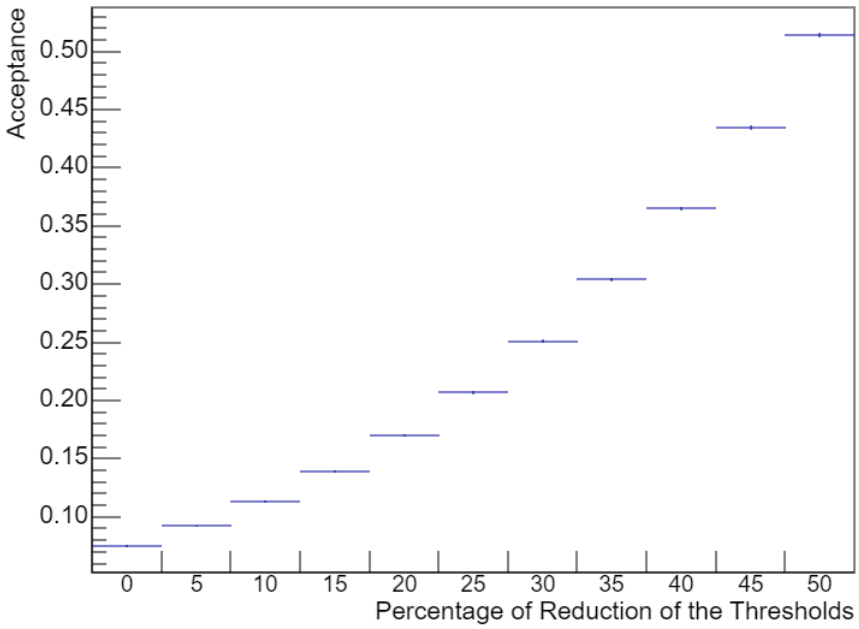


Figure 2.11: Percentage of acceptance for the trigger requiring one b-tagged jet with $p_T >$ 175 GeV and one b-tagged jet with $p_T > 60$ GeV, as a function of the percentage of reduction of the thresholds.

for the second trigger, and we can notice lower values for the acceptance, ranging from 1% for the (250 GeV, 125 GeV) momenta thresholds, to about 12% for (87.5 GeV, 30 GeV)

momenta. The difference in the acceptances for the two triggers is due to the fact that the Higgs bosons will decay into low-momenta b-jets, hence the higher acceptance when the p_T requirements are lower thanks to the additional flavour-tagging requirement. The

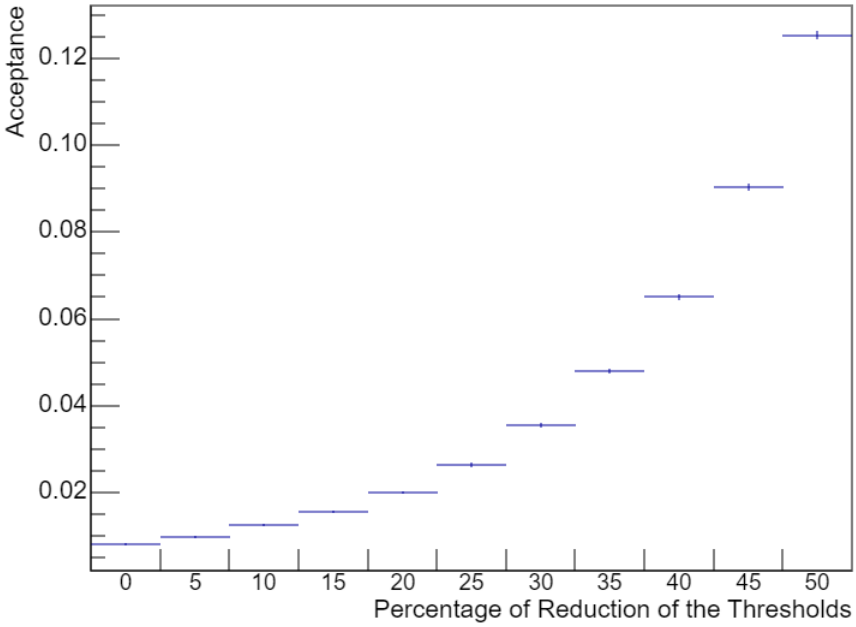


Figure 2.12: Acceptance for the trigger requiring two jets with $p_T > 250$ GeV and one jet with $p_T > 125$ GeV, as a function of the percentage of reduction of the thresholds.

plots that I have just shown hint to the possibility of lowering the momentum thresholds of the triggers that we have selected in order to extend the physics reach when searching for the $HH \to 4b$ process. We also have to remember that this study is still incomplete, since there's also the need to study the behaviour of the background when the p_T values are lowered, since the deciding parameter is the ratio between the $HH \to 4b$ events and the background events passing the selection.
Anyway the results seen in figure 2.12 and 2.11 motivate us to study the possibility of lowering the thresholds of the trigger. But lowering the thresholds causes an increase in the rate of the events and consequently in the workload of the CPU, so that it becomes necessary to lighten the CPU resources occupied by other processes within the HLT in order to allow a higher rate of events.
In chapter 3 we'll see in detail that the most CPU-consuming task at the HLT level is the reconstruction of tracks, and in the following work we'll propose a new approach to achieve a faster tracking, so that lowering the thresholds may become possible for the HLT in ATLAS.

2.3.2 Run 3 trigger and $HH \to 4b$

In chapter 1 we have looked at the new strategy adopted at the HLT during Run 3 [14], where a fast reconstruction of the tracks and a fast b-tagging algorithm are used to enact a pre-selection, hence decreasing the rate of events which will be more carefully analyzed in the following stage of the HLT.
This strategy allows to decrease the rate of events in such a way that the CPU resources required for the full reconstruction and the b-jet trigger finding procedure can now be handled by the HLT farm. In Run 3 a relatively high L1 seed of 8 kHz can be processed, but it is important to verify that the gain achieved thanks to the high L1 rate is maintained despite the fast b-tagging pre-selection, which will decrease the $HH \to 4b$ acceptance.
First of all, we list the trigger requirements: for the L1 selection three central jets ($|\eta| < 2.5$) with $E_T > 15$ GeV are needed, with the leading one having $E_T > 15$ GeV. Then for the pre-selection at least four jets with $p_T > 20$ GeV are necessary, of which two have to be b-tagged by fastDIPS. Finally four jets are required such that $p_T > \{80, 55, 28, 20\}$ GeV of which two b-tagged.
Testing the Run 3 HLT, it is found that the pre-selection reduces the rate up to a factor 10 [14], necessary to perform full-scan tracking, while the acceptance of $HH \to 4b$ decreases of only 2-4% because of the pre-selection. Raising the p_T thresholds or adding other trigger selections would have caused a way larger loss. In table 2.2 we find the details of the results obtained testing the Run 3 HLT.

Pre-Selection WP	Rejection Factor	Relative Acceptance Loss
85%	~ 5	0.98
80%	~ 10	0.96

Table 2.2: Results for the Run 3 HLT. Here the Working Point is referred to the fixed efficiency of the fast b-tagging algorithm.

Chapter 3

Tracking: From ATLAS to Toy

In this chapter we focus on the real-time reconstruction of tracks, explaining in detail the algorithms that compose the tracking process in the ATLAS HLT. I'll first talk about the HLT tracking workflow during Run 2, describing the Data Preparation, Fast Tracking and Precision Tracking algorithms, used to reconstruct, from the ID data, the tracks that are then utilized by the different items of the HLT to take the trigger decision.
I'll also explain what is meant with multiple stage tracking, and how the detector areas exploited at each stage change in order to obtain the most efficient reconstruction.
We'll then discuss about the timing of the various HLT algorithms and we'll notice that the most time consuming processes are related to track reconstruction, showing a strong dependence on the pile-up of the events processed, as the timing increases in a more than linear fashion as a function of $\langle\mu\rangle$. I'll also describe possible CPU-consumption mitigation strategies, such as the one already adopted in Run 3 b-jet triggers, which introduces an early rejection, based on the results of a ML-based b-tagging algorithm.
Anyway, this kind of optimization will not be enough to cope with the harsh detector occupancies expected during the High-Luminosity phase of LHC, where the average number of pile-up collisions will increase up to values around 150, almost three times the level experienced during Run 2, causing a dilation of tracking timing.
Thus we propose a ML-based algorithm trained to filter out space-points produced by pile-up tracks, in order to reduce the possible combinations checked by the tracking algorithm.
In the second half of the chapter I'll talk about the toy events that will be used to train the ML algorithm to filter out pile-up tracks, which are simulated at this stage by randomly generated hits. In order for these random hits to resemble the density found in physical jets, I compute the densities of hits found in Monte Carlo simulations of $t\bar{t}$ jets and I set these values for the generation of the toy events. I then introduce the concept of hitmap, which is a representation of the hits registered in the layers of our generated detector, where the coordinates are changed from (x, y) to (layer, ϕ) in order to obtain a data format that can be fed to the neural network with the goal of filtering out the random noise.
Finally I'll discuss how the adopted data representation in terms of hitmaps may affect the reconstruction of the transverse momentum p_T of the track.

3.1 Tracking in ATLAS

In chapter 1 we have already briefly introduced the real-time reconstruction of tracks within the HLT, as well as the strategies used for both Run 2 and Run 3.
In this chapter we analyze more carefully each step of the tracking workflow, trying to understand the importance of this process within the HLT.

3.1.1 HLT tracking during Run 2

During Run 2 the track reconstruction within the HLT mainly consisted in a two-stage approach, composed of:

1. Fast Tracking
2. Precision Tracking.

In the first stage we have a fast reconstruction of the tracks using trigger-specific pattern recognition algorithms, and then a first selection of the relevant events, while in the second stage the tracks are reconstructed again, with the use of more complex algorithms which utilize the information coming from the previous stage.
We have already shown the complete workflow in figure 1.12, where we can also notice the Data Preparation step, which has the goal to convert the data coming from the ID into space points, and the Hypothesis algorithms, which apply a trigger selection to the events.
We now look in more detail each step of the chain, starting with the Data Preparation: a single particle passing through the ID activates clusters of pixels and SCT strips, with multiple units detecting a fraction of the charge deposited by the passing particle.
The first step in the HLT is to operate a clusterization from the raw data, and to compute a space point for each cluster, which will be then used to reconstruct the track of the particle. The space point is obtained with a weighted average of the position of each activated pixel or strip, where the weights are the charges detected by each unit. For the SCT the space point is obtained combining the measurements from the two layers of strips.
In dense environments multiple particles may be crossing the detector in adjacent pixels, so that the hits caused by different charged particles will be reconstructed as a single cluster. We can see in figure 3.1 an illustration of these clusters, which are called merged clusters.
In order to avoid confusion I now clarify the concepts of hit, cluster and space point:

- Hit: a region of the detector corresponding to a single pixel that has registered a fraction of charge due to the passing of a charged particle.
- Cluster: a collection of nearby hits. It can be caused by the crossing of a single particle or multiple particles passing close by.
- Space Point: geometrical point in the detector obtained with a weighted average of the hits in the cluster. Out of the three it's the more precise estimate of the crossing point of the particle.

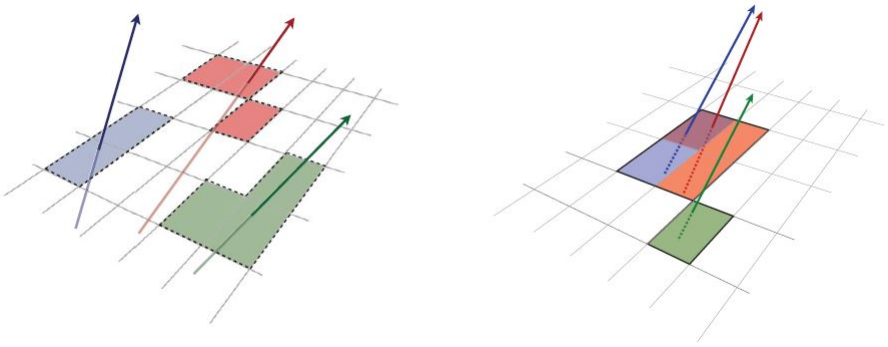


Figure 3.1: Examples of single-particle clusters (left) and merged clusters (right).

To achieve a better time efficiency, the Data Preparation is in most cases performed only for the RoIs obtained from L1, and is performed also for the TRT data only if the event has already been accepted by the fast tracking stage.
The Fast Track Finder (FTF) receives as input the space points within the RoIs and reconstructs track candidates with fast algorithms. The FTF prioritizes efficiency over purity, because the precision tracking stage will apply a further selection, selecting the most likely track between the candidates.
The FTF enacts a pattern recognition, searching for triplets of points, or track seeds, which may belong to the same track. The resulting track candidate also needs to be compatible with the interaction region along the beam line, so that a maximum of $|d_0|$ of 10 mm is allowed. The track seeds are then extended to the remaining layers, in order to find additional space points for the track. Finally the duplicate tracks are removed and a fast Kalman filter is applied to the preliminary tracks.
The precision tracking stage takes the FTF tracks as inputs and applies offline algorithms adapted to run online in the trigger. The tracks are also extended into the TRT detector, in order to find hits at larger radii and improve the momentum resolution of the track. A track fit is then performed, hence finding the final track candidates.
What we have described thus far is referred to as single tracking stage, because both the fast and the precision tracking are run in a single RoI. For example, the electron trigger follows this procedure, with a first selection applied to fast tracks and a further use of the trigger on the precision tracks.
Triggers that look for other particle signatures, like for example the τ and the b-jet triggers, used instead a multistage tracking approach. In the first stage the fast tracking was performed in a narrow RoI, with a full width of 0.2 for both η and ϕ, but extending along the full beam line range, with $|z|$ <225 mm. In the second stage the RoI is wider, measuring 0.8 for both η and ϕ, and along z is now centered on the position of the primary vertex identified in the first stage, limited to $|\Delta z|$ <10 mm.
The tracks obtained from this second stage of fast tracking are then used for the precision tracking, as we have previously described.

This approach allows to rapidly identify the z position of the primary vertex that we are interested to reconstruct, so that we can then investigate in wider angular regions around it, as we can see in figure 3.2.

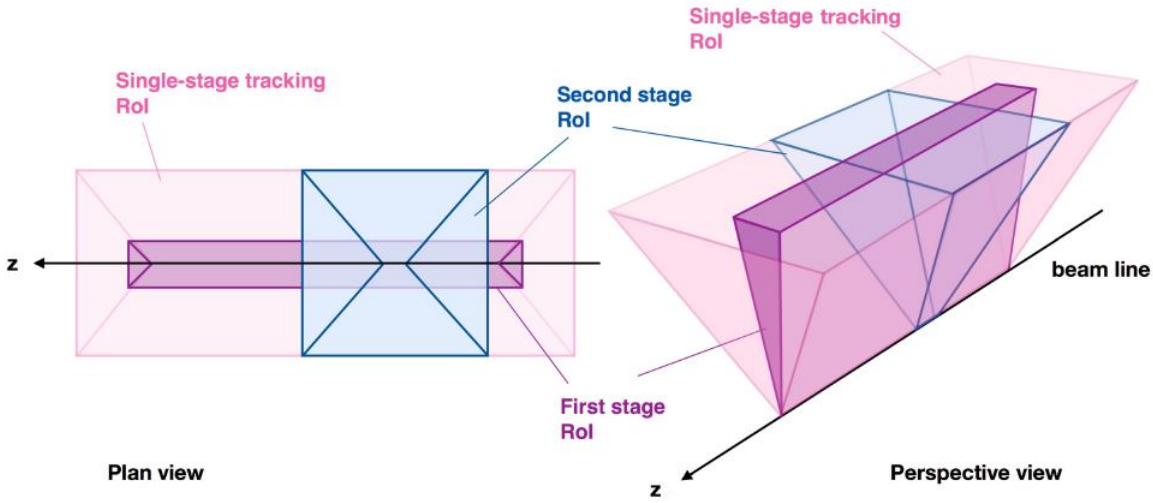


Figure 3.2: Comparison between the RoIs used for single-stage tracking and multi-stage tracking.

For the fast tracking of the b-jet trigger, in the first stage the narrow RoIs are built from jets found at L1 with E_T >30 GeV, and then the wider RoIs are built around the z position found by the primary vertexing algorithm. Subsequently b-tagging algorithms are applied on the precision tracks, and then the trigger selection takes place.
When there is an overlap between different RoIs, in order to avoid processing the same region multiple times, different RoIs are merged in a single super-RoI, which is used for the data preparation. A schematic view of super-RoIs is found in figure 3.3.

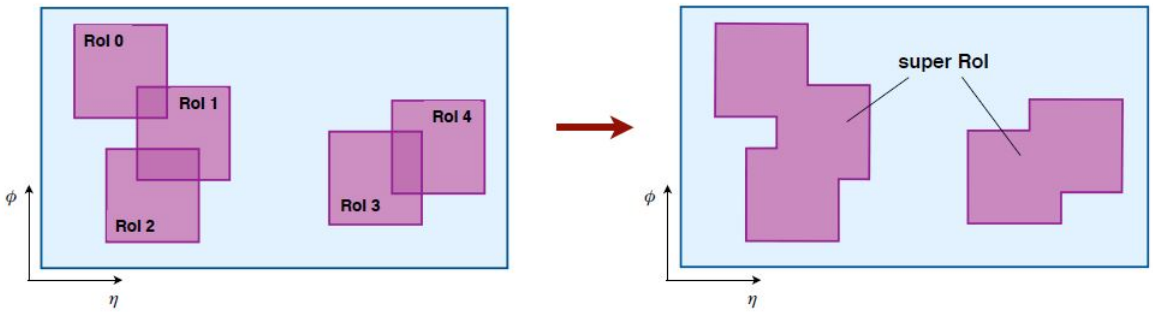


Figure 3.3: View in the $\eta - \phi$ plane of a super-RoI.

3.1.2 Tracking Timing during Run 2

The trigger system operates in real-time, while the collision are happening within the detector; it is then necessary for the online selection of the events to respect the required latency, avoiding the overflow of events and the loss of data. In this context we now look at the timing of the main HLT tracking components, which have been measured online during Run 2 [13].
Different trigger items will have different timings, indeed while the tracks for the electron trigger are obtained with a single stage approach, the tracks needed for the tau, muon and b-jet trigger use multi-stage tracking, causing the increase of the timing.
We now investigate the average execution time per event at the HLT, looking also at the dependence of the timing on the average pile-up value $\langle\mu\rangle$, obtained in the collisions. A higher value of the pile-up will correspond to a higher density of hits detected in the ID, which will then impact the algorithms at HLT level.
On the left of figure 3.4 we see the timings for the pixel and SCT clustering and for the space-point identification; as we expected the data preparation for the pixel detector is the one requiring the higher timing, with values from 10 ms up to 30 ms for events with high pile-up. We also notice a more than linear dependence on $\langle\mu\rangle$ for the pixels, while for the SCT and the space-point finder the behaviour is linear.

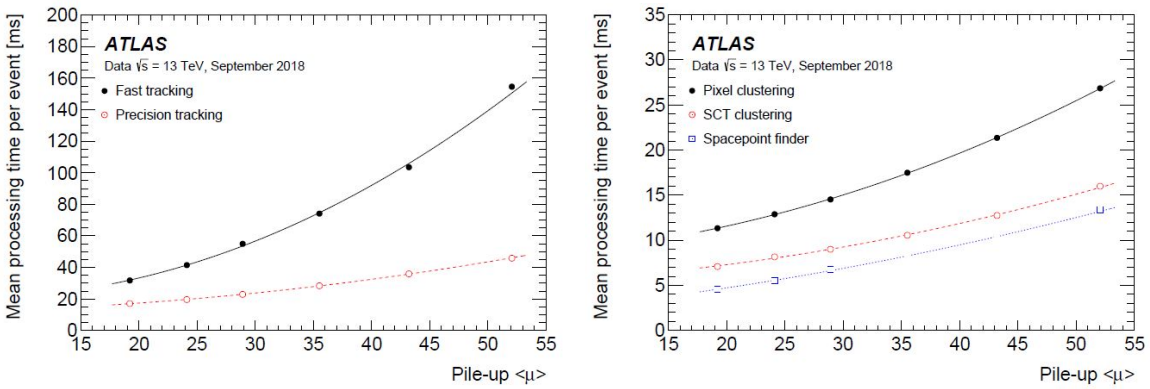


Figure 3.4: Processing time as a function of the pile-up, for the algorithms in the HLT during Run 2.

On the right of figure 3.4 we now examine the timings for the fast and precision tracking; while the precision tracking shows a linear dependence on the pile-up value, with timings in the range 20-40 ms, the fast tracking timing is instead a strongly more than linear function of $\langle\mu\rangle$, ranging from 30 up to 160 ms.
First of all, the FTF is more time consuming than precision tracking, whose processing can already profit from the background reduction operated by the previous step, thus facing a much lower number of space point combinations.

The power-like shape of the FTF timing is explained by the combinatorial nature of the algorithms used in the fast tracking stage; during the process of track seeding, the algorithm looks for a series of three space points which are compatible with being produced by the same track, and doing so, all the combinations of space points need to be checked. Increasing the space-point density, with higher $\langle\mu\rangle$, a way higher number of combinations will arise, hence slowing down the process.
This power-like dependence on $\langle\mu\rangle$ for the fast tracking is a critical weakness for the trigger system; indeed the average values for the pile-up expected at the HL-LHC phase, will be in order of 150-200, meaning that the HLT software used in Run 2 would not be able to manage the volume of data acquired with high luminosity.
As we'll see in the following, in Run 3 a new strategy has been adopted for some trigger selections, which anyway utilizes basically the same combinatorial algorithms for FTF.
In this thesis I want to propose a new approach to this problem, which makes use of an ML-based algorithm; the idea is to obtain an ML program which receives the space points obtained in the ID and filters out the hits due to pile-up tracks within the RoI, so that ideally only hits from a specific interesting event remain, hence decreasing in a major way the number of points to be processed by the FTF algorithms.
The ML algorithm that we'll use, will receive inputs with fixed dimensions, and so the timing to filter out pile-up hits won't depend on the hit density of the event, as we'll show in chapter 5.
In order to test this new approach we start with training the ML model on toy events that we generate independently, as I'll have explained within the end of this chapter.

3.1.3 HLT tracking for b-jet triggers during Run 3

In chapter 1 we have briefly described the change applied to the HLT b-jet selection for Run 3, and in chapter 2 we have already seen how this change has an impact on the acceptance of the $HH \to 4b$ channel. We now look more closely to the new strategy, describing it into detail.
Tracking was the most CPU and time consuming process during Run 2 data taking; if a way to speed up this process was found, it would be possible to rededicated a considerable amount of resources to lower the p_T thresholds of the jet selection. This would help expanding the physical reach of the experiment, as we have seen in chapter 2.
With this goal in mind, the HLT strategy has been changed for Run 3, introducing an early selection that utilizes a b-tagging algorithm, which allows an early rejection and significantly reduces the event rate. While the workflow of the HLT has changed, the algorithms used for fast tracking and precision tracking are basically left unchanged from Run 2.
First of all, jets are reconstructed from energy deposits in the calorimeters, which are referred to as EMTopo jets, and fast tracking is run on the super-RoI defined at the previous step. From fast tracks the algorithm fastDIPS is deployed, identifying b-jets. As we touched on in chapter1, fastDIPS machine learning-based algorith which is optimized to perform fast b-tagging on the low quality tracks available at trigger level.

Events can be rejected either at the jet reconstruction or at the b-tagging steps, with the goal of substantially reducing the rate of events without discarding relevant physical processes. For the events selected, fast tracking is run again, this time on the full range of the pixel and SCT layers, and the jets are reconstructed again, this time as "Particle-Flow" (PFlow) jets. Particle Flow is an approach that combines information from the ID and the calorimeters, to achieve a better event reconstruction.
For the jets that pass the E_T and η selections, precision tracking is performed and the following steps are same that were performed in Run 2.
In figure 1.13 we have already seen a comparison between the HLT strategies used in Run 2 and in Run 3, which illustrates what I have just explained. In figure 1.13 we can also notice that a low rate of very energetic events is allowed to skip the preselection and go directly to the full scan FTF.
Before inserting this new workflow in the HLT, multiple tests have been conducted on Monte Carlo simulated events, to determine the optimal width in ϕ and η for the RoIs in the early rejection with fastDIPS, as well as the p_T cut value for the tracks reconstructed. The working point chosen is to use angular widths of 0.3 and accept the tracks with $p_T \geq$1 GeV, obtaining only a small loss in the performance for a significant gain in the CPU used, which is 30% of what would be used with the full ID acceptance.

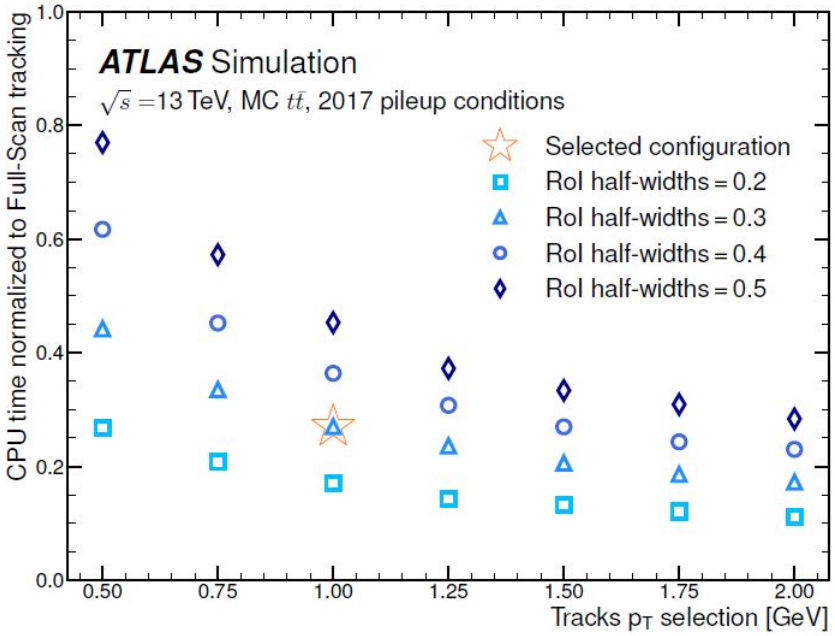


Figure 3.5: Fraction of the CPU utilized with respect to the full scan tracking for different working points in η, ϕ and p_T.

3.2 DiTTo

In paragraph 3.1.2 we introduced the object of analysis in my thesis work, a ML algorithm that could filter out the space-points due to pile-up tracks, in order to significantly reduce the number of points to be processed by the fast tracking algorithms.
In order to evaluate the performance of this algorithm, we start by testing it on simple events, that we simulate using a toy generator, where we can easily control the parameters of the events created. While in ATLAS an event is the data detected as the result of the interaction of two bunches of protons, in this context an event corresponds to the data obtained from an idealized jet, which will be composed of signal hits and noise hits. If the study of the algorithm acting on such toy events leads us to good results, we can then move on and apply the algorithm to full simulations of ATLAS jets. In this thesis we'll focus on the study of tracking algorithm performance on toy events, leaving the analysis on ATLAS events for further developments.
The simulation code that we'll be using to generate the toy events, and for which I took part in the development, is called DiTTo, which stands for Differentiable Tracking Toy. The word Differentiable is there because the long-term plan is to utilize Differentiable Programming tools in the track reconstruction process; we'll talk about this topic later, as one of the prospects for further development of my work.
The toy model that we utilize to generate the events, separates the hits into two classes:

SIGNAL HITS: DiTTo generates ideal curved tracks originating from the beam pipe, and then simulates the detection of the signal hits. There is the possibility to choose the distribution for many key parameters like the number of tracks per event, their transverse momentum and angular variables.
The coordinates of the signal hits correspond to the exact intersection between the track and the cylindrical layers, ignoring the disposition and the discrete geometry of the Pixels or the Silicon strips. Later in this section we'll give more details about how the intersection is computed.

NOISE HITS: in order to simulate pile up tracks we initially add to the events some random background hits, in a way that the hits in different layers are **not correlated** with each other. In this way the noise hits aren't due to actual pile up tracks, but they're generated randomly using angular densities set by the user; this simple description will allow us, anyway, to extract information about the denoising power of the algorithm, making a first step towards more realistic and complicated models.

DiTTo is a generator of 3D events, but for our purposes we want the input of the algorithm to be a flat image, so we focus on 2D projections of the events, either on the x-y or on the z-r plane, with $r = \sqrt{x^2 + y^2}$.
Within the DiTTo environment, I developed a simple tool for the display of the events, in figure 3.6 an event with a single signal track plus noise is shown, looking at the projections on the x-y plane.

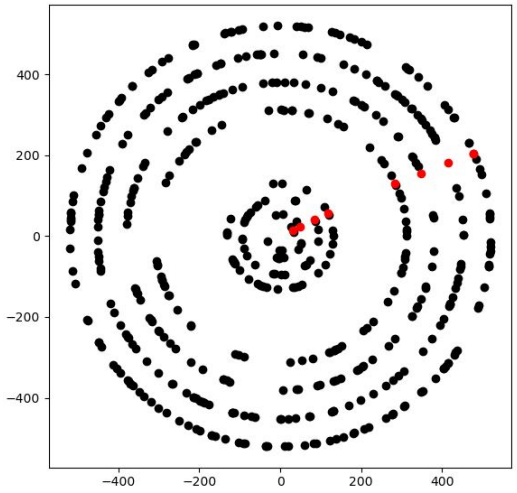


Figure 3.6: Signal hits are displayed in red while background hits are in black.

We now give a brief explanation of the settings of DiTTo, so we'll go through all the settings that allow to generate different samples. We can divide the configurations of DiTTo into two categories:

Detector Configurations : the geometry of the detector is set to have cylindrical symmetry, in this section we can decide the number of the layers and also the value of their radii, so that DiTTo can be adapted to a wide range of cylindrical tracking detectors. Furthermore we can set the range for both the z coordinates and the ϕ values of the hits generated, which is useful to create samples with hits only in limited regions of the detector.
We can also set the average angular density of noise hits to be produced for each layer, as well as the corresponding RMS, so that we can control the noise production regardless of the ϕ interval that was set. Once the number of noise hits in a layer is decided, the ϕ value of each hit will be extracted randomly within the ϕ interval.
Finally we have to take into account the possibility that when a particle goes through a layer it may not be detected, so we introduce the layer efficiency in order to simulate this property of the detector.
The efficiency is the probability that a hit in the layer will be considered detected, otherwise we get rid of this hit. We can set different efficiencies for each layer.
There's also the possibility to ignore every hit that goes through a layer, making it inactive. This option is useful to test the robustness of the reconstruction process with respect to detector failures, as we'll see in chapter 5.

Tracks Configurations : here we set the details of the generation of the tracks that will lead to the detection of the signal hits. We introduce the possibility to generate many tracks in a single event, and we can also obtain a sample with a variable number of tracks per event, selecting the minimum and the maximum number of tracks desired. The p_T interval for the tracks can be decided, as well as the probability distribution used to extract the p_T values; for example we can use a flat distribution or a negative exponential distribution.
We can also set the average value and RMS for the longitudinal and transverse impact parameters z_0 and d_0, and the z coordinate of the primary vertices.

3.2.1 Hit Density from $t\bar{t}$ jets

Now that we have introduced DiTTo in full generality, we want to simulate events within a geometry inspired by the ATLAS Barrel ID, with 4 layers playing the role of the Pixel detector and the other 4 layers as the SCT. So the simulated detector will have 8 layers with the following configuration:

layer	layer 0	layer 1	layer 2	layer 3	layer 4	layer 5	layer 6	layer 7
radius[mm]	34.1	54.2	95.0	131.0	312.0	381.0	452.0	520.0

Table 3.1: Radius of each layer of the detector.

Now that we have adapted the geometry of the DiTTo detector, we also tune the configurations relative to the noise hits production, so that they approximately match a specific physical example that we would like to investigate.
The long-term goal of this work is to obtain a ML algorithm that could take jets as inputs and perform track reconstruction within them. To do so, we set the angular density of random noise hits generated by DiTTo, in order for it to match the average angular density of hits found in Montecarlo (MC) simulations of jets coming from $t\bar{t}$ events, with the pile-up distribution of Run 2.
The angular density of hits is dependent on the layer of the detector and also on the average number of primary vertices produced in the beam crossing; we indeed expect that the density will increase if the pile-up interaction multiplicity increases.
Examining a MC sample of $t\bar{t}$ events we plot the distribution for the number of primary vertices (nPV) per bunch crossing , as we can see in figure 3.7. In figure **??** instead I have plotted the average angular density of hits per layer, for the $t\bar{t}$ jets in the same sample.

The MC simulated samples are representative of a full year of data-taking (2022), and so we observe a wide distribution for the number of primary vertexes, since during the year the beams have gone through different settings.

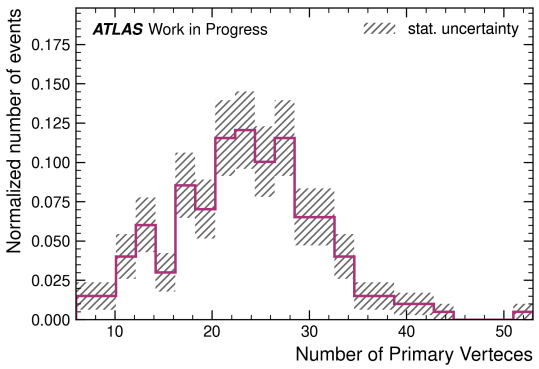


Figure 3.7: : Distribution of the number of primary vertices per bunch crossing.

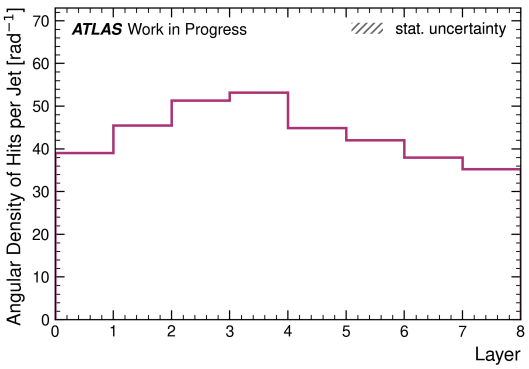


Figure 3.8: Average angular densities of hits for each layer

But we don't want to replicate the properties measured over such a long period of time, we would like for our simulated events to be representative of briefer periods, where the distribution of the number of primary vertices will be way narrower.
In order to build an event generator with this property we study how the average number of hits in each detector layer depends on the average number of primary interactions. To do so, I scan different values for the number of primary vertices ν, taking the events within the window $[\nu - 2, \nu + 2]$ and, for the jets contained in this events, I compute the angular density of hits in each layer.
In figure 3.9 we see the angular densities for each layer of the ATLAS detector (indicated with different colors), calculated for the jets in the $t\bar{t}$ sample. We notice that the angular density increases with respect to the number of primary vertices and also that the innermost layers are the ones with the highest angular density.

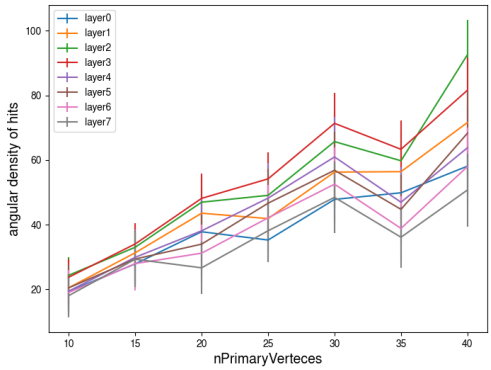


Figure 3.9: Angular density of hits vs nPV.

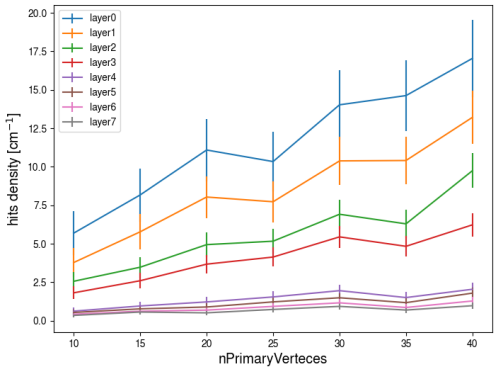


Figure 3.10: Linear density of hits vs nPV.

In figure 3.9 I've also plotted the "linear" density of hits, as the number of hits in the layer divided by the length of the arc subtended by the jet, which we calculate as $l = r\Delta\phi$, with r the radius of the layer and $\Delta\phi = 0.8$, since the jet occupy $\phi \in [-0.4, 0.4]$. We can observe again that the densities increase in a more or less linear fashion with respect to the number of PVs and that, as we could expect, the linear density is higher for the layers closer to the beam pipe.
As we said earlier, DiTTo requires the angular density of noise hits for each layer, so we have to decide a value for ν and then plug into DiTTo the angular densities for each layer that we obtain in the graph 3.9.
In figure 3.11 we show two different events, one with angular density of noise hits compatible with an interaction with 10 primary vertices and another one compatible with 40 primary vertices.
As we'll be doing for most of our studies, we focus on the ϕ interval of $[-0.4, 0.4]$, since we'd like to apply our algorithms to jets, and by now, we keep the d_0 value for the tracks set to zero. For both pictures the signal track has $p_T = 1$ GeV.

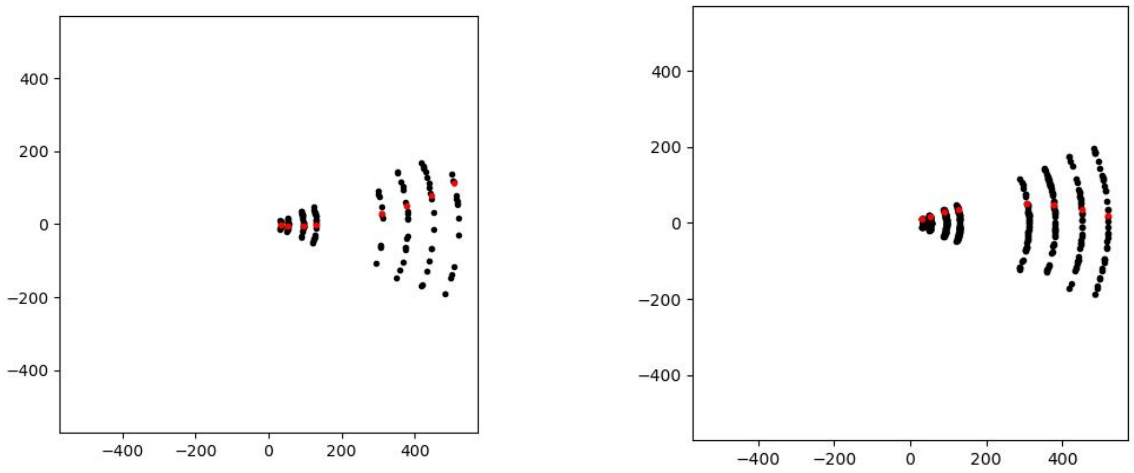


Figure 3.11: Image of a simulated jet in events containing respectively 10 (left) and 40 (right) primary vertices.

3.2.2 From Hits to Hitmaps

Our algorithm will be a CNN and so its input will be an image; we could feed to the network an image like figure 3.6, but we can enact a change of coordinates that can lead to a more natural representation of the hits. So we decide to plot a new set of variables:

$$(x, y) \longrightarrow (\text{layer}, \phi) \tag{3.1}$$

In the following figure we take the same events showed in figure 3.11 and we plot them with the change of coordinates that we have just described:

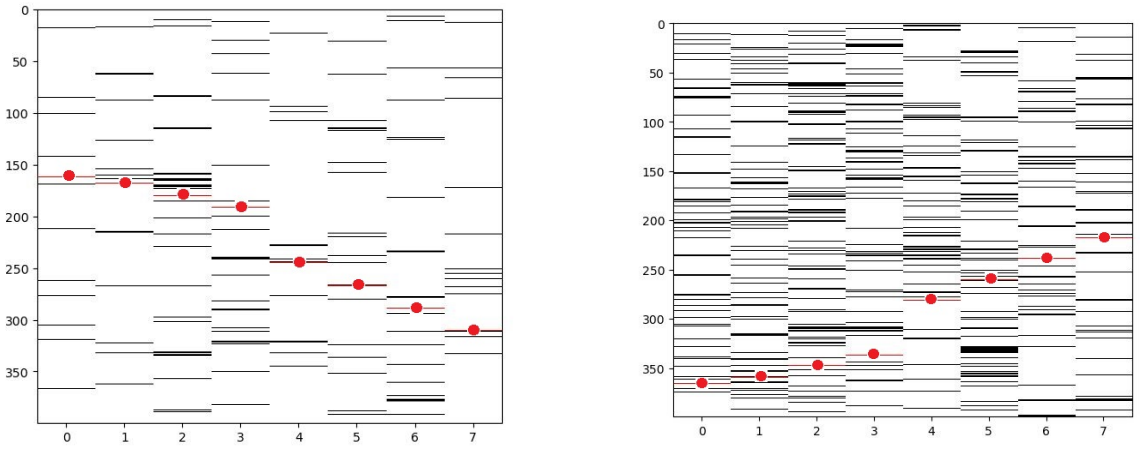


Figure 3.12: Display of a hitmap associated to 10 PVs (left), and another one to 40 PVs (right), plotting layer and ϕ. Red dots indicate the signal hits.

We observe that in the new coordinates each hit can be identified with a single pixel of image while this isn't clear in the x-y view.
Images like the one shown in figure 3.12 will be the inputs for the CNN, and we will refer to them as **hitmaps**. This kind of representation is directly produced within the DiTTo environment, where both the training and the testing dataset will be generated.
When we act the trasformation in equation 3.1 we introduce an element of discontinuity, indeed looking at the table 3.1 we notice that while the distances between the first 4 layers and between the last 4 are fairly the same, the distance between layer 3 and layer 4 isn't portrayed properly on the layer axis.
Looking at a track with p_T of 1GeV we see the following hitmap:

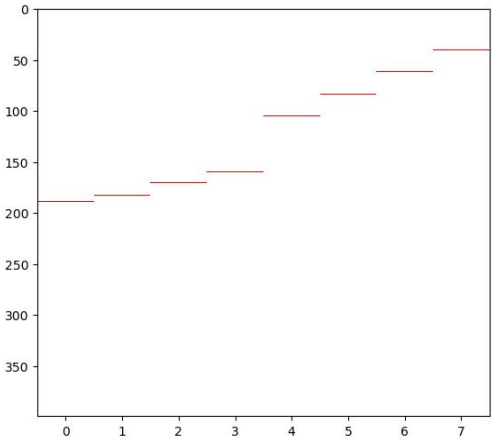


Figure 3.13: Signal hits due to a track of 1 GeV

We are basically seeing two straight lines separated by the discontinuity jump we were talking about, and since the spacing between the layers is bigger for the SCT, the slope of the line will be bigger in this section.
This consideration is something to keep in mind when we will study whether our network can learn such track pattern shapes or not.

3.2.3 Binning along ϕ

While the binning for the layer coordinate is trivial, the same thing cannot be said about the binning for ϕ, because the choice of the width of the bins is arbitrary and so it's up to us to decide.
Indeed also the bin width along ϕ, which we call $\phi_{\mathbf{bin}}$, is a parameter for the generation of events, and through out this thesis we'll be using:

$$\phi_{\text{bin}} = 0.002 \text{ rad} \tag{3.2}$$

We have opted for this value mainly for practical reasons, in order to avoid having images too big and also semi-empty; we could study how the performance of the algorithm depends on the choice of the bin width along ϕ, but we will consider this question for further developments of this work, and proceed utilizing $\phi_{\text{bin}} = 0.002$ rad.
Introducing the binning for the ϕ coordinate, we are describing a discrete structure for our simulated detector. But, while in the ATLAS detector, further layers are provided with more pixels or silicon strips than the previous ones, for our purpose we are binning all layers with the same ϕ_{bin} so that we obtain a constant number of ϕ bins and we're able to format the view of the detector in a rectangular image.
With this description we are basically building a detector with pixels that increase their width along the ϕ direction for further layers; to avoid confusion we will refer to these "ideal pixels" as bins. We would like to compare the size of these bins with the actual pixels and silicon strips present in the ATLAS detector: in table 3.2 we have computed the width w of the bins in the DiTTo detector by simply doing $w = r\,\phi_{\text{bin}}$ for each layer, and also reported the values for the sensor widths in ATLAS, seen in chapter 1.

layer	layer 0	layer 1	layer 2	layer 3	layer 4	layer 5	layer 6	layer 7
DiTTo[μm]	68.2	108.40.0	190.0	262.0	624.0	762.0	904.0	1040.0
ATLAS[μm]	50.0	50.0	50.0	50.0	80.0	80.0	80.0	80.0

Table 3.2: Comparison of the widths of the ATLAS pixels and the effective DiTTo pixels.

We notice that only in the first layer (the IBL for ATLAS) we have comparable widths, and for further layers the pitch of the DiTTo bins becomes way larger.

If the width of the DiTTo bins would have been lower than the pitch for the ATLAS pixels, then we would have had to add signal hits also for the neighboring bins along the ϕ direction, because otherwise we would have simulated a detector with higher resolution then what is available in ATLAS. But now we have checked that this never happens, so it's not necessary to apply this correction.
Furthermore it could have been interesting to do some studies regarding the smearing of the signal hits, considering that in some cases the hit can be reconstructed in the wrong pixel, and we could have looked if the performance of our algorithm would be degraded if the smearing was added.
But looking at table 3.2 it's clear that the influence of the smearing would impact at most the first layer, because the hit will be wrongly reconstructed at worst in the neighboring pixels, and since every DiTTo bin after layer 0 has at least double the pitch than the ATLAS pixels, the smearing would barely have an effect on the image reconstructed. So we can skip altogether the study about the presence of the smearing.
The binning along ϕ introduces though a limit for the maximum observable p_T value; indeed after this threshold the image shown in the hitmap will simply be a straight line and so it will not be possible to tell apart tracks with higher p_T.
We want to obtain this threshold, and we start by computing the coordinates of the hit of a particle with transverse momentum p_T with the layer of the detector of radius r.
In the simulation of tracks performed by DiTTo we assume a uniform magnetic field of intensity $B = 2$ T, and the particles associated with the tracks will have charges $q = \pm 1e$, so that we can use the known formula:

$$p_T = 0.3 \; B \; R \tag{3.3}$$

where p_T is expressed in GeV, B in Tesla and R, which is the radius of curvature of the track, in meters.
Considering a particle that comes out of the beam line with exit angle $\phi_0 = 0$, since at this stage we are assuming the transverse impact parameter d_0 to be zero for all tracks, we can express the trajectory of the particle with the equation of a circle with radius R and center in $(0,R)$.
We now solve the following system:

$$\begin{cases} x^2 + y^2 = r^2 \\ x^2 + (y - R)^2 = R^2 \end{cases} . \tag{3.4}$$

Solving the system for x and y we get:

$$\begin{cases} x = r\sqrt{1 - \frac{r^2}{4R^2}} \\ y = \frac{r^2}{2R} \end{cases} . \tag{3.5}$$

This is the same calculation that is done within DiTTo to obtain the coordinates of the track hits, the only step remaining is to rotate (x, y) by the exit angle ϕ_0, that in general will be different from zero.

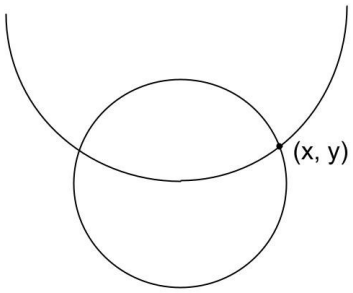


Figure 3.14: View of the intersection of the circles.

Getting back to our calculation, we now compute:

$$\tan(\phi) = \frac{\mathrm{y}}{\mathrm{x}} = \frac{1}{\sqrt{\frac{4R^2}{r^2} - 1}} \tag{3.6}$$

If we now plug $\phi = 0.002$, the value of the bin width, and we use the radius of the last layer, r = 520.0 mm, we get the value of p_T such that the track is visualized on the hitmap as a straight line, obtaining:

$$p_{T,\,\mathrm{thr}} = 39 \text{ GeV} \tag{3.7}$$

So a track with $p_T > p_{T,\,\mathrm{thr}}$ and passing by the bin center in the first layer will span at most over the immediately adjacent bin in phi along its trajectory, leading to a negligible pT measurement capability.
This threshold does not impact the ability of the algorithm to distinguish the signal tracks from the background, on the contrary a track with $p_T > p_{T,\,\mathrm{thr}}$ will be depicted as a straight line on the hitmap and we expect that the algorithm will easily recognize this pattern.
As a possible extension of the work presented in this thesis, we also might want to build an algorithm to infer the value of the transverse momentum of a track, in order to use it for trigger purposes. As we said, $p_{T,\,\mathrm{thr}}$ will be the limit value for which we can hope to reconstruct a truthful value for p_T, since there no way for the algorithm to distinguish tracks with higher momentum.
We can anyway demonstrate that the obtained $p_{T,\,\mathrm{thr}}$ value isn't actually a limitation for typical tracks belonging to b-jets; indeed only a very small percentage of the tracks reconstructed in $t\bar{t}$ events have $p_T > p_{T,\,\mathrm{thr}}$. In figure 3.15 we can see the distribution relative to the p_T of the tracks contained in the $t\bar{t}$ sample that we have used thus far and we can confirm that only a negligible fraction of the tracks will have momentum grater than $p_{T,\,\mathrm{thr}}$. So the choice of $\phi_{\mathrm{bin}} = 0.002$ rad, which implies the above momentum threshold of $p_{T,\,\mathrm{thr}} =$ 39 GeV, will not impact in a significant way our ability of reconstruct the momentum of the tracks reconstructed within the detector.

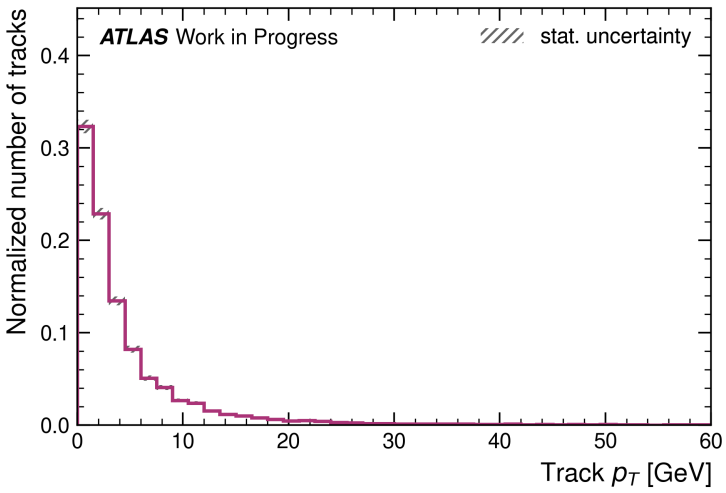


Figure 3.15: Distribution of the tracks in the MC $t\bar{t}$ sample in the range [0 GeV, 60 GeV].

Chapter 4

Machine Learning and CNNs

In the last chapter I presented the problem of the more than linear dependence of the fast tracking timing with respect to the average pile-up of the events, and I also proposed a new approach that could help to mitigate this issue. This approach involves the use of a ML-based algorithm and so, in the following chapter, I will introduce the concepts necessary to fully understand and utilize a ML algorithm, more specifically a Convolutional Neural Network (CNN).
I'll start with a general presentation of Machine Learning, introducing the concepts of task, experience and performance measure in this context. We'll also talk about unsupervised and supervised learning, and define what is the training of an algorithm and how this is connected to the loss function. With the example of the Linear Regression, we'll introduce the importance of the weights of an algorithm, and we'll minimize the loss in this simple case. I'll later discuss the meaning of Underfitting and Overfitting, and how they are related to the capacity of the algorithm, which is the ability of generating complex functions to fulfill the task.
The filtering algorithm that we would like to adopt at trigger level is a Deep Learning algorithm, in which the information travels through multiple layers of neurons, which we'll understand discussing the example of the Perceptron. For Deep Learning algorithms, also called neural networks, it isn't generally possible to minimize analytically the loss function, so that gradient-based learning is used, which consists in iteratively computing the gradient of the loss and then move in the weights space in order to find the configuration corresponding to the minimum of the loss.
With the tools just listed we'll then introduce CNNs, neural network algorithms that receive images as inputs, and with the use of the convolutional layer obtain new representation of the input, where patterns and correlations between different parts of the image can be observed. We'll look closely at this kind of layer and define the kernel of a convolution, which scans through the input image and recasts it in a different shape.
After having defined the Max Pooling and the Upsampling layer, we'll be ready to head onto the next chapter and utilize the algorithm, knowing every layer that is employed to build the architecture of the model that we'll use.

4.1 Introduction to Machine Learning

Artificial intelligence or AI is nowadays an increasingly important field of research, with applications both for scientific purposes and in everyday life.
AI and machine learning (ML) are often used interchangeably, but ML is a subset of the broader category of AI.
With AI we refer to the ability of computers to emulate human behavior and perform tasks in real-world settings, while with ML we refer to algorithms that are able to perform such tasks without being explicitly programmed, learning directly from the interaction with the data. The contents of this chapter are mainly inspired by [24].
In order to define more precisely what we mean with learning we introduce:

- **the task** T is the goal or class of goals of the algorithm,
- **the experience E** describes the interaction between the algorithm and the training data,
- **the performance measure** P is the metric utilized to evaluate the abilities of the algorithm in carrying out the task T.

A machine learning algorithm is said to have learned if the performance in executing the task T, as measured by P , has improved after an experience E.
We now give a more detailed description of what we mean with task, experience and performance measure.

4.1.1 The Task

The task is the goal of the algorithm, the objective that we are aiming to learn or predict. ML allows us to face problems that may be too difficult to solve with deterministic algorithms.
We now list some common machine learning tasks:

- **Classification**: given k categories of objects, the goal of this task to receive an input object and decide to which of these categories it belongs to.
  To do so the algorithm will usually produce a function:

$$f : \mathbb{R}^n \longrightarrow \mathbb{R}^k \tag{4.1}$$

  where $\mathbb{R}^k$ refers to the k probabilities of the object to belong to the k-th class, the one with the highest probability is chosen as the category fit for the input.
  There are many examples of this task, such as spam email filtering or object recognition.

- **Regression**: in this task the program is asked to compute a number as the output, which describes a certain feature of the input.
  This time the algorithm will produce this function:

$$f : \mathbb{R}^n \longrightarrow \mathbb{R}. \tag{4.2}$$

  An example of this task is the prediction of the expected claim amount that an insured person will make given their risk factors, or predicting the price of a house given the features of the house.

- **Transcription**: here the algorithm is asked to observe some kind of data and then offer a textual description of it.
  This can be used for example for captioning, giving an image as input and obtaining a brief text that describes the content of the image. Another use is speech recognition, where the program is provided an audio and then outputs a sequence of character describing the words spoken in the recording.

- **Anomaly Detection**: in this case the program goes through a series of objects in the data and learns to recognize unusual objects compared do the baseline pattern. An example of an anomaly detection task is credit card fraud detection.

- **Denoising**: in this task, the algorithm receives a corrupted object $\tilde{x} \in \mathbb{R}^n$, obtained with an unknown corruption process from a clean object $x \in \mathbb{R}^n$, and the goal is to obtain an output object as close as possible to x.
  This task is applied to image reconstruction, in order to obtain a neater version of the image, for example by removing noise components introduced by the sensor electronics.

Later in chapter 5 we will apply denoising to the tracking problem, so we'll have a chance to explore this task in more detail.

4.1.2 The Experience

Machine learning algorithms can be broadly categorized as supervised or unsupervised by what kind of experience they are allowed to have during the learning process.

Unsupervised learning algorithms experience a dataset where data points are described by $x \in \mathbb{R}^n$ with n is the number of features available to describe x. This dataset does not come with labels, meaning that no additional information is attached to the data points.
Unsupervised algorithms learn useful properties about the inherent structure of the data, and this information can be used to perform tasks such as clustering. Indeed the model learns a metric that describes whether two inputs are similar, and applying this metric it's able to group different inputs together.

Supervised learning algorithms instead interact with data points that have a label or a target associated with them: for example in a classification task each object in the training sample has a label with the category that we want to predict.
Observing several examples of input vectors x and their associated value or vector y, supervised algorithms learn to predict the output $\hat{y}$ from a generic input x.

We mention also **Reinforcement Learning** algorithms, where the algorithm itself interacts with an environment, so that at every action enacted by the program there is a feedback from the environment that changes how the system will act in the future.

In the application of ML to my thesis work, we'll focus on supervised learning algorithms. For this class we now describe the learning process or "Training". First of all, we describe an algorithm as a function $f(\mathbf{w}, \mathbf{x})$, where $\mathbf{w}$ are the sets of weights or parameters that control the behaviour of the model, and $\mathbf{x}$ is the generic input. The model interacts with a training dataset acting on it and obtaining the corresponding outputs:

$$\hat{\mathbf{y}}^{\text{training}} = f(\mathbf{w}, \mathbf{x}^{\text{training}}) \tag{4.3}$$

From these results the "Loss Function" is computed, measuring the discrepancy between the output $\hat{y}$ and the target value y, and computing the average for each element in the training dataset.

$$L_{\text{tot}}(\mathbf{y}^{\text{train}}(\mathbf{x}^{\text{train}}, \mathbf{w}), \hat{\mathbf{y}}^{\text{train}}) = \frac{1}{N} \sum_{i=1}^{N} L(y_i^{\text{train}}(x_i^{\text{train}}, \mathbf{w}), \hat{y_i}^{\text{train}}) \tag{4.4}$$

The model is now updated, changing the weights from **w** to **w'** in order to reduce the loss; this process is iterated until the loss is sufficiently low and the model has learned the task. The loss is a function which penalizes bad predictions; if the model's prediction is perfect, the loss is zero, otherwise the further we are from a good prediction, the bigger the value of the loss will be. The algorithm will try to minimize the loss, so the definition of it is a fundamental step in the setup of a training, because it has to reflect the task that we'd like the system to perform. We'll later get more into detail about how the weight are updated.

4.1.3 The Performance Measure

We need a metric to evaluate how well the algorithm is executing the task, this metric is the performance measure.
We are usually interested to see how well the program performs on new data, different from what was used for the training, since this determines the performance of the algorithm when deployed on input data.
We therefore evaluate the performance measures using a testing dataset that is separate from the data used for the training. Indeed the algorithm must be able to perform well also on previously unseen inputs, this property is called "generalization". The "generalization error", also called test error, is nothing more than the expected error on a new input coming from the testing dataset, and expresses a measure for the performance of the algorithm.
For a classification task a good measure could be the accuracy of the prediction, defined as the average probability of the algorithm classifying an input into the correct category, so that we assign a value from zero to one to the performance of the program.
For a Regression task instead we would like to measure how far the prediction of the model is from the desired output value, and so we could use the Mean Squared Error (MSE) calculated between the outputs of the model and the labels assigned to the inputs.

$$MSE(\mathbf{y}^{\text{test}}, \hat{\mathbf{y}}^{\text{test}}) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i^{\text{test}} - y_i^{\text{test}})^2 \tag{4.5}$$

4.1.4 Linear Regression

Since the discussion we had so far has been quite abstract, let's try to make things more concrete with an explicit example of a simple ML algorithm: Linear Regression.
As the name implies, the task at hand is a Regression problem, so starting from an input $\mathbf{x} \in \mathbb{R}^n$ we want to obtain $\hat{y} \in \mathbb{R}$ as an output.
For Linear Regression the output is a linear function of the input, so we write:

$$\hat{y} = \mathbf{w}^{\mathrm{T}}\mathbf{x} \tag{4.6}$$

where $\mathbf{w} \in \mathbb{R}^n$ is the vector of parameters. Tuning these values we change the behaviour of the algorithm.
Interpreting each component x_i as a feature of the input, $\mathbf{w}$ can be seen as a set of weights that determine the importance of each feature in making the prediction of $\hat{y}$.
Having defined the task T we now need to decide a measure performance P; as we already said, for regression problems we can choose the MSE of the output of the model on the testing set, and identify it with the Loss of the model.

$$L_{\text{tot}}(y^{\text{train}}, \hat{y}^{\text{train}}) = MSE(y^{\text{train}}, \hat{y}^{\text{train}}) \tag{4.7}$$

When the algorithm experiences the training set $(x^{\text{train}}, y^{\text{train}})$ we want it to find a set of parameters $\mathbf{w}_{\text{min}}$ that minimize the Loss of the model:

$$\mathbf{w}_{\text{min}} = \arg\min_{\mathbf{w}} L_{\text{tot}}(\mathbf{w}). \tag{4.8}$$

For the example of Linear Regression we easily find such weights:

$$\nabla_{\mathbf{w}} \mathfrak{L}(y^{\text{train}}, \hat{y}^{\text{train}}) = 0 \implies \mathbf{w} = \frac{\mathbf{x}^{\text{train}^{\mathrm{T}}} y^{\text{train}}}{x^{\text{train}^{\mathrm{T}}} x^{\text{train}}}. \tag{4.9}$$

We have found the set of weights that performs the best given the model expressed in equation 4.6.
Linear regression is often used to refer to a slightly different model, which contains an additional term b, the bias:

$$\hat{y} = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b \tag{4.10}$$

But if we redefine $\mathbf{w}$ and $\mathbf{x}$ as:

$$\bar{\mathbf{w}} = (b, \mathbf{w}) \quad \bar{\mathbf{x}} = (1, \mathbf{x}) \tag{4.11}$$

we can find the solution for this affine model rewriting the previous one.
Linear Regression is of course a very simple algorithm and it lacks the power to manage complex tasks, we'll need to utilize more evolved models in order to do so.

4.1.5 Underfitting and Overfitting

Under the assumption that the training dataset and the testing dataset are produced with the same data generating process, so that they come from the same probability distribution, we can say that for an untrained model, with randomly selected weights, the training error and the testing error will be the same.
In the previous example of Linear Regression, we trained the model minimizing the training error $L(y^{\text{test}}, \hat{y}^{\text{train}})$ while the test error is $L(y^{\text{test}}, \hat{y}^{\text{test}})$.
But once the training starts, since the model utilizes the training dataset to adjust its parameters, the expected training error will be less or equal to the testing error.
We can now define the following concepts:

Undefitting occurs when the algorithm isn't capable of obtaining a sufficiently low test error, so that the task is executed poorly.

Overfitting occurs when the gap between the training error and the testing error becomes too large, more precisely when the training error keeps decreasing and the test error starts instead to grow, meaning that their derivatives have opposite sign.

Whether a model will overfit or underfit is controlled by its **Capacity**, or the ability of the algorithm to generate a wide variety of functions.
Models with a high capacity will overfit, because in order to reduce the loss they have the possibility to adapt to the peculiarities of the training dataset, which do not generalize to the testing dataset.
We now give a direct example: let's imagine to generate a training dataset selecting random

values of x and calculating $y = x^2$, so that the points belong to a parabola.
Considering three different models:

$$\hat{y} = b + wx, \tag{4.12}$$

$$\hat{y} = b + w_1 x + w_2 x^2, \tag{4.13}$$

$$\hat{y} = b + \sum_{i=1}^{9} w_i x^i. \tag{4.14}$$

Even though model 4.13 and 4.14 aren't linear functions of the input, they are still linear in the parameters, and so we can still solve as we did in 4.9.
Figure 4.1 shows the plots of the functions that we obtain with the new weights:

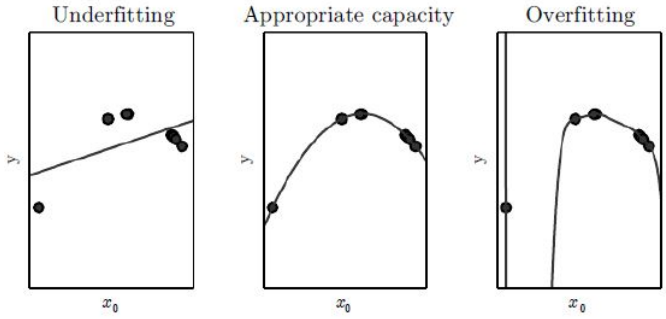


Figure 4.1: Example of underfitting (left), and overfitting (right), where the model adapts to the peculiarities of the dataset.

We observe that model 4.12 isn't able to represent the curvature of the data and so it **underfits**. While model 4.13 fits the data and will generalize well for the testing dataset, model 4.14 also fits the data but the system has adapted to this particular set of training data and will create a large testing error; the model has **overfitted**.
While model 4.12 doesn't have enough capacity for the task, model 4.14 has to much and ends up overfitting.

4.2 Introduction to Deep Learning

In this chapter we'll give a review on some concepts that are involved when we talk about Deep Learning, for instance we will talk about the Perceptron, with the goal of understanding the structure of a Neural Network.
Later we'll get a little bit more technical introducing the Gradient-Based learning and what we mean when we talk about optimizers.

4.2.1 The Perceptron

The **Perceptron** is the building block of neural networks, and historically one of the first steps towards Deep Learning.
To explain the structure of the Perceptron we can we can go back to the linear regression example, where an additional step is added; on the output of the linear combination of the inputs and the parameters is then applied a non-linear function called **Activation Function**. Using equation 4.10 the function defined by the Perceptron is:

$$\hat{y} = g(f(\mathbf{x})) = g(b + \mathbf{w}^{\mathrm{T}}\mathbf{x}) \tag{4.15}$$

where g is the activation function and f is the linear function seen in the context of Linear Regression. The bias term b is a trainable parameter just like the components of $\mathbf{w}$, and is often indicated as w_0:

$$\hat{y} = g(w_0 + \sum_{i=1} x_i w_i) \tag{4.16}$$

Figure 4.2 shows a graphical representation of a Perceptron:

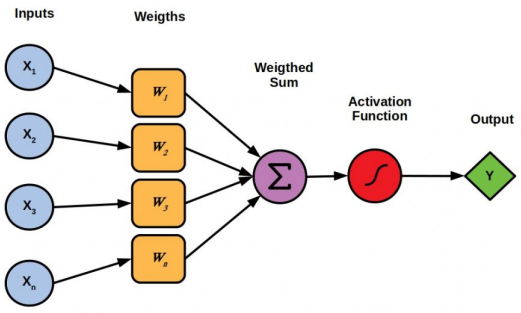


Figure 4.2: Illustration of a perceptron, with the combination of inputs and weights and then the activation function.

The Activation function plays a fundamental role because it introduces non-linearities in the system hence increasing the capacity of expression of the model.

Let's take a look at some of the most used activation functions:

- the ReLU, or Rectified Linear Unit, is one the most popular and efficient activation functions.
  It is defined as:
  $$g(z) = \max(0, z) \tag{4.17}$$
  As we can see in figure 4.3 the output of the ReLU is always positive.

- The Sigmoid function is defined as:
  $$g(z) = \frac{1}{1 + e^{-z}} \tag{4.18}$$
  This activation function may be useful when the desired output $\hat{y} \in (0, 1)$.

- The Hyperbolic Tangent, defined as:
  $$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{4.19}$$
  This may be useful when $\hat{y} \in (-1, 1)$.

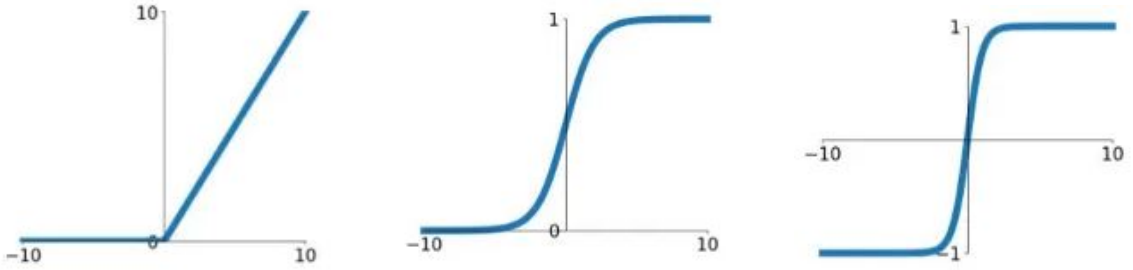


Figure 4.3: Here we see in order, the ReLU, the sigmoid and the hyperbolic tangent.

We can also have multi-output perceptrons, where the added nodes will have dedicated weights, as summarized by this formula:

$$\hat{y}_i = g(w_{0,i} + \sum_{j=1} x_j w_{j,i}) \tag{4.20}$$

The perceptron is still a very basic algorithm which doesn't have the capacity to face complex tasks, but if we stack multiple perceptrons we obtain what is called a Feedforward Neural Network or Multi-Layer Perceptron (MLP).

4.2.2 Feedforward Neural Networks

If we compose two perceptrons, taking the output of a multi-output perceptron and making it the input of the next one, we obtain a more complex structure called **Feedforward Neural Network**.
Considering two multi-output perceptrons which act on the respective inputs with the functions $f^{(1)}$ and $f^{(2)}$, then the output of the system will be:

$$\hat{y}_i = f^{(2)}(f^{(1)}(\mathbf{x})) \tag{4.21}$$

Each function $f^{(i)}$ of the network defines a **layer**, and the number of layers is called the **depth** of the model.

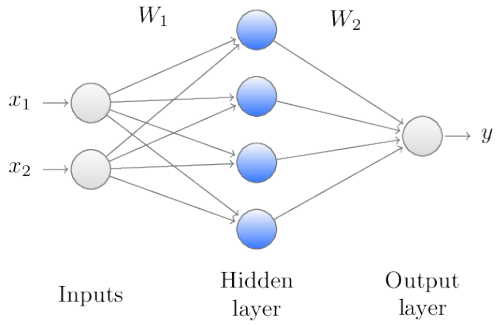


Figure 4.4: Neural network with two layers.

The word "feedforward" means that the information flows straight from the input to the output, without feedback connections in which some outputs are fed back into the model.
The term "neural" comes instead from an analogy with neuroscience, indeed each component of the representation offered by the hidden layer can be seen as a sort of **neuron**, in the sense that it receives input from many other units and computes its own activation value to be passed to the next layer.
In this thesis we'll only deal with Feedforward Neural Networks, and from now on we'll refer to them simply as Neural Networks.
In figure 4.4 we identify the inputs, the output layer and the **hidden layer**; this layer contains a new representation of the input in a different dimension, obtained through the action of the weights and the activation functions of the first layer.
Once the training is completed we expect this representation to portray more clearly some features of the input, which will be useful for the task at hand.
When the model presents more than one hidden layer, we talk about **Deep Neural Networks**.
The goal of most neural networks is to approximate a certain function $f^*(\mathbf{x})$, which corresponds to the action on the inputs that we need in order to perform the task perfectly; changing iteratively the values of the weights $\mathbf{w}$, the goal of the network is to obtain a function $f(\mathbf{x}, \mathbf{w})$ as close as possible to $f^*(\mathbf{x})$.

So, in the context of neural networks, what we mean by "learning" is modifying the parameters of the model, until we get to the optimal weights for our task; this translates mathematically in the need to calculate the **gradient** of very complicated functions.

4.2.3 Gradient-Based Learning

We recall the concept of **Loss**, that we defined in the previous chapter as a performance measure which assumes higher values the more the output of the model is far from the desired target.
We also saw the example of Linear Regression, where the optimal weights of model could be computed solving:

$$\nabla_{\mathbf{w}} \mathfrak{L}(\mathbf{y}^{\text{train}}(\mathbf{x}^{\text{train}}, \mathbf{w}), \hat{\mathbf{y}}^{\text{train}}) = 0 \tag{4.22}$$

where, as we saw in equation 4.4, the Loss can be written as an average over the training data of a Loss calculated for each data point. It won't be as easy for neural networks; in this case the model of the network is a highly non linear function, so that we won't be able to solve equation 4.22 in a closed form.
Furthermore when we deal with Deep Neural Networks the number of parameters can easily be in the order of millions, complicating the problem even further.
What we would like to do is to build an iterative algorithm to find the optimal weights, in order to do so we can still utilize the gradient of the Loss, using the technique known as **Gradient Descent**.
First of all we set all the weights of the model to random and small values $\mathbf{w}_{\text{start}}$, calculate the loss and then compute its gradient in $\mathbf{w}_{\text{start}}$.
We know that the gradient of a function points towards the direction of fastest increase, so if we go in the opposite direction we'll hopefully be closer to a minimum of the Loss.
So we can update the weights of the model in this way:

$$\mathbf{w}' = \mathbf{w}_{\text{start}} - \eta \, \frac{1}{N} \sum_{i=1}^{N} \nabla_{\mathbf{w}} L(\mathbf{y}_i^{\text{train}}(\mathbf{x}_i^{\text{train}}, \mathbf{w}), \hat{\mathbf{y}}_i{}^{\text{train}}) \tag{4.23}$$

where η is the **Learning Parameter**, a positive value that determines the size of the step in the parameters space and can be decided a priori by the user.
Looking at formula 4.23 we notice a problem: the gradient of the loss function is calculated over the entire training dataset, which contains N data points, with N that could easily be in the order of millions.
This means that at each step of the training we would have to perform $\mathcal{O}$(N) operations, which would increase enormously the computational time.
An approach to solve this issue is to calculate the gradient of the loss for only one data point $\mathbf{x}^{\text{random}}$, chosen randomly from the training dataset.
This algorithm is called **Stochastic Gradient Descent** (SDG), and the updated weights are:

$$\mathbf{w}' = \mathbf{w}_{\text{start}} - \eta \, \nabla_{\mathbf{w}} L(\mathbf{y}^{\text{random}}(\mathbf{x}^{\text{random}}, \mathbf{w}), \hat{\mathbf{y}}^{\text{random}}) \tag{4.24}$$

In this way we are reducing drastically the computational cost of the algorithm, even if the direction taken in each step of the training will be less precise, since it is obtained only considering one data point.
A compromise between full Gradient Descent and SDG is **Batch Gradient Descent**, where at each step we sample a **batch** of data from the training dataset, and we use this sample to compute the gradient of the loss.
In this case the update rule for the weights is:

$$\mathbf{w}' = \mathbf{w}_{\text{start}} - \eta \frac{1}{m} \sum_{i=1}^{m} \nabla_{\mathbf{w}} L(\mathbf{y}_i^{\text{train}}(\mathbf{x}_i^{\text{train}}, \mathbf{w}), \hat{\mathbf{y}}_i{}^{\text{train}}), \tag{4.25}$$

where with m we indicate the size of the batch.
If m is big enough to represent the full distribution of data points, we expect this gradient to be a good approximation of the one calculated over the entire dataset.
When every data point in the training dataset has been chosen as part of a batch we say that an **epoch** has occurred and we restart the process of sampling batches.
The number of epochs is a measure to determine how long the training will be, meaning how many times each data point in the training set will have contributed to the update of the weights.

To evaluate if a model is **Overfitting** or **Underfitting**, what is usually done is to divide the training set into data actually used for the update of the weights and the **evaluation set**, which is usually around the 10% of the dataset.
At each epoch the generalization error is computed on the evaluation set and compared to the training error.
What is obtained is a plot like the one shown in figure 4.5: when the evaluation error starts to grow larger than the training error we know that the model is adapting to the peculiarities of the training set, hence overfitting.

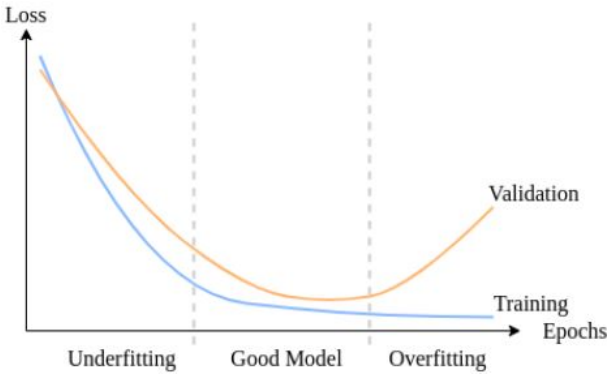


Figure 4.5: Example of overfitting, with the test error and the training error having derivatives with opposite signs in the last epochs.

4.2.4 Momentum and Adaptive Learning

We now introduce the **Momentum** method, designed to accelerate the learning: the idea of this algorithm is to consider also the gradients of past moves when we update the weights. To do so a variable $\mathbf{v}$ is introduced, which plays the role of the velocity with which the updating weights move in the parameters space, and the influence of the past gradients decreases exponentially the further we go back to previous steps.
To determine how quickly thede contributions decay, the hyperparameter $\alpha \in [0, 1)$ is introduced, and the update works like this:

$$\mathbf{v}_t = \alpha \mathbf{v}_{t-1} - \eta \frac{1}{m} \sum_{i=1}^{m} \nabla_{\mathbf{w}} L(\mathbf{y}_i^{\text{train}}(\mathbf{x}_i^{\text{train}}, \mathbf{w}), \hat{\mathbf{y}}_i^{\text{train}}) \tag{4.26}$$

$$\mathbf{w}' = \mathbf{w} - \mathbf{v}_t \tag{4.27}$$

where with t we indicate the number of the training step.
In figure 4.6 the black arrows indicate the steps that would be taken with simple gradient descent, while the red path going faster towards the minimum, is obtained with the momentum algorithm.

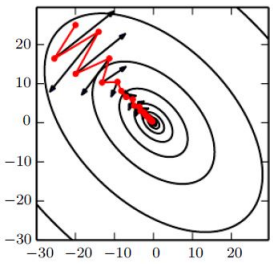


Figure 4.6: Effect of the momentum algorithm on the training.

The momentum algorithm accelerates the training, but there is still room for improvement. An idea could be to allow the possibility for the learning parameter to change along the steps of the training, creating what are called **Adaptable Learning Rate** algorithms. We now focus on some of these algorithms, since we'll be using them in practice, later in the thesis.

Adagrad [25]: we define the quantities $G_{j,j}$ as:

$$G_j = \sum_{\tau=1}^{t} g_{\tau,j}^2 \tag{4.28}$$

where $g_{\tau,j}$ is the partial derivative of the Loss with respect to the parameter w_j, calculated at step τ.

Then the weights are updated following the rule:

$$w'_j = w_j - \frac{\eta}{\sqrt{G_j + \delta}} g_j \tag{4.29}$$

where η is the learning rate and $\delta = 10^{-7}$ is added for numerical stability.
So Adagrad adapts the learning rate for each parameter, dividing by the square root of the sum of the square of the partial derivatives of the loss along each step of the training.
However the accumulation of all the squared gradients from the beginning of the training results in an excessive decrease of the learning rate, which basically causes the training to stop.

RMSprop : descends directly from **Adagrad**, and tries to cure the problem of the learning rate going to zero.
The solution proposed by RMSprop is to change the gradient accumulation into an exponentially weighted moving average, in a way that gradients further in the past will eventually contribute zero.

Adam [26]: this algorithm can be seen as a combination of RMSProp and momentum, indeed both the accumulation of the gradients (within what we called the velocity $\mathbf{v}_t$) and of the square of the gradients is done with an exponentially decaying average.

Later in the thesis I will utilize both RMSprop and Adam for the same network, and evaluate the difference in the performance.

4.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are neural networks where at least one of the layers is a convolutional layer, specialized in processing images, or anyway data with a grid-like topology.
CNNs are one of the main tools utilized for Computer Vision, which is an increasingly relevant field of research interested in developing new techniques to extract information from images, in order to perform Classification, Image Retrieval, Detection, Segmentation of Images and many more tasks.
Our goal is to obtain an algorithm that could filter an image containing both signal hits and noise hits, in order to obtain an output image containing only signal hits and a CNN may be what we are looking for.
Let's now talk about the Convolutional Layer and its properties, as well as some other kinds of layers that will make up the CNN that we'll be using to denoise the events produced by DiTTo.

4.3.1 Convolutional Layer

Since convolutions are found in many fields, first of all we clarify the meaning in this context: given two functions $f(t)$ and $g(t)$ the convolution operation, denoted with an asterisk, is defined as:

$$(f * g)(t) = \int f(a)\ g(t-a)\ da \tag{4.30}$$

Assuming now that the functions f and g are defined on the integers, we can define a discrete convolution:

$$(f * g)(x) = \sum f(a)\ g(t-a) \tag{4.31}$$

If we consider both f and g to be 2D images, we can define a convolution product acting on both dimensions:

$$(I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)\ K(m,n) \tag{4.32}$$

In the context of convolutional networks the first argument of the convolution is the input of the layer, the second argument is called **kernel** and the output can be referred to as the **feature map**.
While this product has the nice property of being commutative, another one is used in practice when implementing convolutional layers, this product is called **cross-correlation** and is defined as follows:

$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)\ K(m,n) \tag{4.33}$$

For simplicity we'll refer to cross-correlation as convolution, and we use again the symbol "$*$" to denote the product.
We now define the **kernel** of the convolution, which is a matrix containing parameters of the model, which will be initially set randomly and then updated during the training.
We now take for example an input image of size N×N, and F×F matrix for the kernel and starting from the upper-left corner of the input, we consider a sub-image containing F×F pixels.
We now combine the sub-image and the kernel multiplying each element of the first-one times the element in the same position in the kernel and then adding all the products, obtaining a number which will be the first element of the output matrix.
This scalar product is equivalent to the usual dot product between vectors, if we think of stretching the matrices in order to form 1D arrays.
We now slide the F×F window of one pixel to the right and repeating the previous operation we again obtain a number that will be the element in the first row and second column of the output.

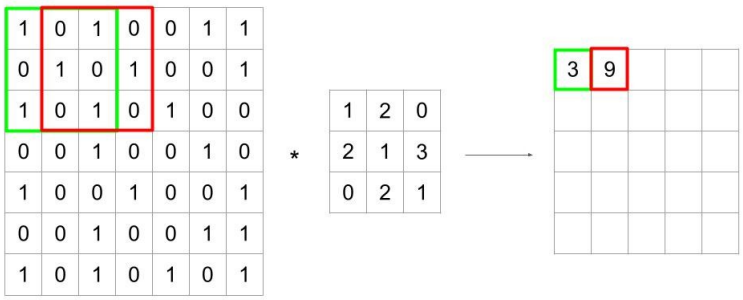


Figure 4.7: Example of the first steps of a convolution with stride 1.

If we iterate this procedure we obtain an (N+F-1) square matrix; we now add to all the elements of the matrix the bias element, thus obtaining the output of the convolutional layer. We observe that the action of a 1×1 kernel would leave the input unchanged.
In the example shown in figure 4.7 we have slid of only one pixel to the right, and after having slid all across the first row we then move from the upper left and slide down again of one pixel. The number of pixel skipped at every iteration takes the name of Stride, and it was set to 1 for the previous example. I now show the result for stride 2 with the previous input and the same kernel:

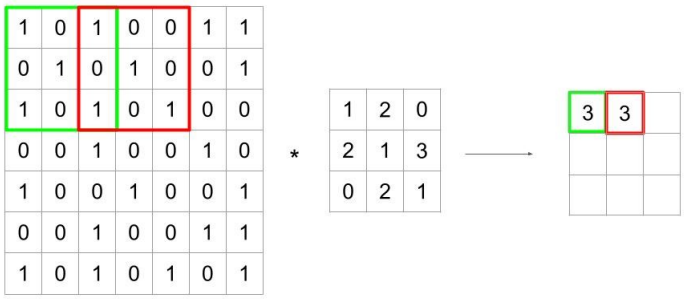


Figure 4.8: Example of the first steps of a convolution with stride 2.

Using Strides>1 has the effect of a further downsampling of the image, reducing in an important way the size of the input.
We notice that the output matrix is now a 3×3 matrix, the output size O can be in general calculated in this way:

$$O = \frac{N - F + 1}{\text{stride}} + 1 \tag{4.34}$$

Looking closely we see that with $N = 7$ and $F = 3$ it's not possible to use stride 3; to solve the problem we can use Zero Padding, adding zeros around the input image in such a way that now the convolution with stride 3 is possible.

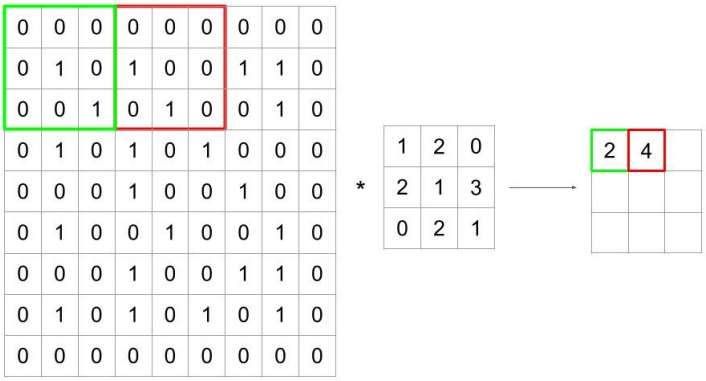


Figure 4.9: Example of the first steps of a convolution with zero padding and stride 2.

Until now we have treated the input image a 2D matrix, from now on we are introducing a third dimension, the **depth** of the image.

This idea doesn't seem so strange if we think about the RGB color model, which is used for Televisions, computers and any kind of image display; within this model every image is composed of three different layers, each layer corresponding respectively to shades of Red, Green and Blue. When the three layers are summed we obtain the image with the full spectrum of colors.
In this context the multiple layers of the image will correspond to different representations of the data flowing through the network, where, once the model is trained, we'll find information related to different features of the input.
Let's consider an input image of shape (32, 32, 3), if we want to apply a convolution using a 3×3 kernel, then we need to apply three different kernels, that we collectively call "filter", so that its shape will be (3, 3, 3). The filter is now composed of 27 parameters, and the convolution will work as follows: each layer of the filter will apply the convolution to the corresponding layer of the input image and the 3 values obtained will then be summed to obtain a single number as the output. If we now slide the window as we did before the output takes shape (28, 28, 1), as we can see in figure 4.10

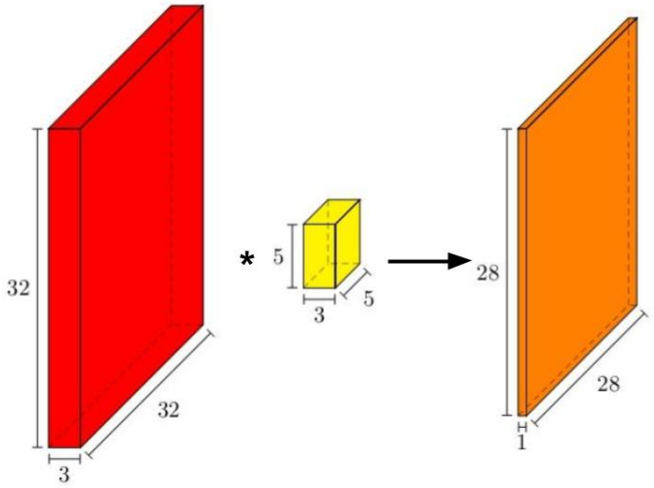


Figure 4.10: Illustration of the convolution operation.

If we use another filter of shape (3, 3, 3) and apply it again to the input we obtain again an output shaped (28, 28, 1); we can stack this image with the one obtained previously to form a comprehensive output, if we imagine to use D filters the output will now have shape (28, 28, D). In general if we use D filters of size (F, F, d) acting on a (N, N, d) image, the convolutional layer will have the following number P of parameters:

$$P = (F \cdot F \cdot d + 1) \cdot D \tag{4.35}$$

We commented earlier that in the context of one layered images a 1×1 kernel acts as the identity on the image, but if the N×N image has depth d and we apply D kernels of shape (1, 1, d) the convolution will provide an output of shape (N, N, D) changing the input.

This operation implements a combination between the elements of different layers, and can be used as a "bottleneck"; when in a network the depth of an image is getting too big, a 1×1 convolution can be applied to return an image with the desired depth.
Fully Connected layers apply a matrix multiplication between the input and the matrix of the parameters, so that each parameter describes the relation between each input unit and output unit.
Convolutional layers present instead **sparse connectivity**, meaning that the units of the output are the result of the combination of a limited number of units in the input, decided by the shape of the kernel. Indeed a kernel with shape (F_1, F_2) is said to have an $F_1 \times F_2$ Receptive Field.
In Deep neural networks though we'll find interactions that involve indirectly larger portions if not the all the input units, so that the model learns both local and global features.

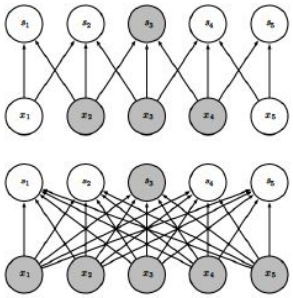


Figure 4.11: Conv. layers connectivity (upper figure) vs connectivity for FC layers.

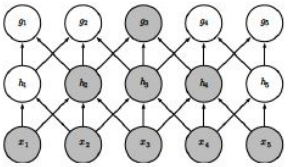


Figure 4.12: Connections for deeper layers.

Convolutional layers are an example of **parameter sharing**, meaning that the same parameters are used for multiple components of the model. Parameter sharing is an advantage because it allows to find shared patterns, to generalize features and it also reduces the number of computations in the model
Because of the parameter sharing in the kernel, convolutional layers present **equivariance to translations**, meaning that if an image is shifted the representation offered by the convolutional layer will undergo the same shift.
Two functions f and g are said to be equivariant when $f(g(x) = g(f(x))$, in this situation f is the convolutional layer and g is the translation of the image.

4.3.2 Max Pooling Layer

Pooling layers are used to downsample the images and obtain a representation of the image with a smaller size, one example is Max Pooling; we first define a matrix, for example of size (2, 2) that will be used to scan the input image. The output will be the image with the max values seen by the scanning matrix at each step, as showed in the following figure:

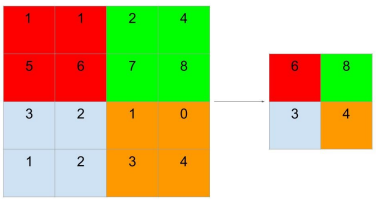


Figure 4.13

The goal of Max Pooling Layers is to produce lower resolution version of the input image which still contains the large or important structural elements, without the fine detail that may not be as useful for the task of the network.
It's also possible to define a rectangular scanning matrix, for example (2, 1), in order to downsample in a different way along the two dimensions.

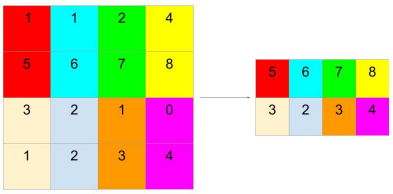


Figure 4.14

Pooling layers do not contain any trainable parameters, and they don't change the depth of the image.

4.3.3 Upsampling Layer

Upsampling layers are used to increase the size of the input image while leaving untouched its depth. Again we have to choose the size of the matrix, for example (2, 1), and each element in the input will be replaced by this matrix, where all its elements are equal to the element of the input.
In the following figure the functioning of the upsampling layer is made clear:

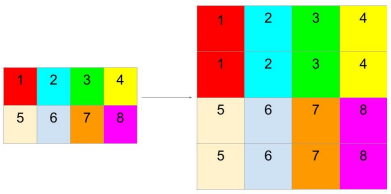


Figure 4.15

Chapter 5

Results and Optimization

The following chapter is dedicated to the analysis of the results obtained with the filtering CNN, utilizing as figures of measure the efficiency and the purity for our hit-selection process, studying also how these depend on the kinematics of the track that originated the signal hits. First of all though we'll talk about denoising autoencoders and we'll be inspired by their architecture to build the model of our CNN.
An appropriate choice for the loss function is a fundamental step in order to obtain a model that executes the task, so we'll justify the decision made for our denoising CNN.
We'll then choose a model for the CNN and we'll use it to explain the process that concludes with the computation of efficiency, purity and the ROC curve. In this thesis we have already found the concept of working point of an algorithm, but we'll define it again in this context. Subsequently we'll look at the dependence of the performance of the algorithm on the p_T of the tracks, the layer on which the hit is found and the ϕ coordinate of the hit. To evaluate the performance of the models, we'll also introduce the Area Under Curve (AUC), and the value of $p_{T,\,\text{turn-on}}$ which will allow us to compare different models.
At this point we'll embark on an optimization process for our algorithm, restraining the possible models to a class parametrized by the starting and ending values of the kernel along the ϕ direction. After having trained all the models in this class we'll compare them using both the AUC and their $p_{T,\,\text{turn-on}}$, to find which of these models is better for the problem at hand. To further investigate the properties of the algorithm we'll train the class of models using both the Adam and the RMSprop optimizers, and again find which one is better for the denoising task.
Finally we investigate how the performance of the algorithm changes when the density of hits in the detector changes, and we look for the dependence of the efficiency as a function of the pile-up associated to the inputs. We would like to have an algorithm that could be robust against the changes of hit density, because our final goal is to utilize this algorithm in the context of HL-LHC.

5.1 Denoising Autoencoders

As we said in the previous chapters, our goal is to build a CNN algorithm that could distinguish signal hits, due to the passing of tracks through the detector, from the random background noise hits, that at this stage simulate pile-up tracks that we would like to remove from the image. In figure 5.1 we see on the left a hitmap with both noise and signal hits, that will be the typical input for our CNN, and on the right we have an image presenting only track hits, which we would like, ideally, to be the output of the algorithm.

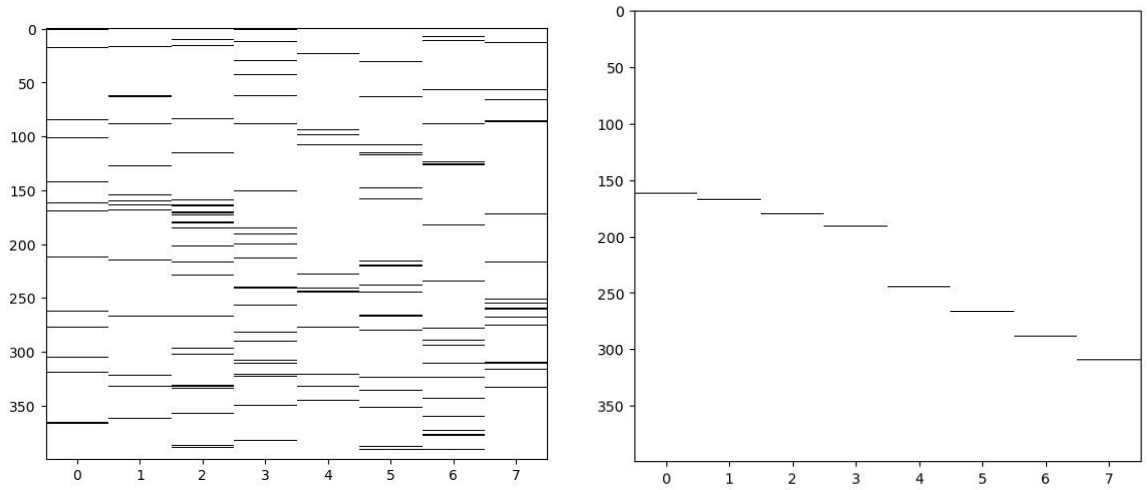


Figure 5.1: Display of the hitmap with both signal and noise hits (left) and the hitmap with only the signal hits (right).

Thinking about it, the task that we want to perform is in analogy with the denoising task, where an algorithm receives as input an image with added noise, and is asked to provide as output the clean image, with the noise removed.

Figure 5.2: Hitmaps with signal and noise hits and then with only the signal hits.

We can see an example applied on the MNIST dataset, which is a sample of images that depict numbers written by hand, and is very often used for examples in the context of ML. In figure 5.2 we see on the left an image from the MNIST sample and then in the middle the same image with gaussian noise added to it; this is the input of the algorithm which, once trained, outputs the image on the right, cleaned from the noise.
There is a particular class of machine learning algorithms called **Denoising Autoencoders** that is indeed used to perform denoising, and so we will be inspired by their architecture to build our model. But first of all let's dive into Autoencoders for a little while, in order to understand which of their characteristics can be useful for us.
An autoencoder is a ML algorithm where the training process aims to learn the input itself, and unlike traditional algorithms where the goal is to predict an output y given inputs x, an autoencoder learns to reconstruct the input x starting from x. This implies that the output of the autoencoder has the same dimensionality as the input.
This task may seem useless since the function that describes the algorithm is simply the identity, but during the training the model is trying to reproduce the training dataset, and doing so some features of the inputs will be learned, and this can indeed be useful for other purposes.
Looking at their architecture, autoencoders present a hidden representation h which takes the name of "code", where we find a new representation of the input, also called latent-space representation, that can be described as a compact summary or compression of the input.

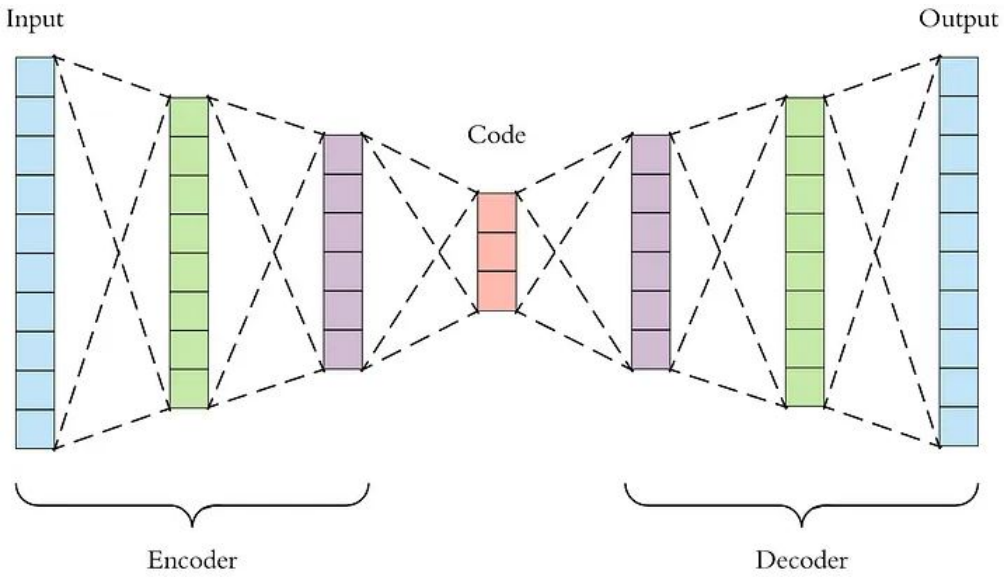


Figure 5.3: Schematic view of the architecture of an autoencoder.

The network may be viewed as consisting of two parts: an Encoder function

$$h = f(x) \tag{5.1}$$

that outputs the hidden layer, starting from the input. We also have a Decoder layer, that has as input the code and produces a reconstruction:

$$r = g(h). \tag{5.2}$$

This architecture is graphically shown in figure 5.3.
As we said earlier, it wouldn't be interesting if the autoencoder learned to simply set g(f(x)) = x for every x, but if we force some constraints on the network, such as decreasing the dimensions of the latent representation with respect to the input, we can discover interesting structure about the data.
When the input goes through the layers of the net and arrives to a representation with a lower dimension, the network is forced to learn a compressed version of the input.
If the input were completely random then this compression task would be very difficult, but if there is structure in the data, for example, if some of the input features are correlated, then this algorithm will be able to discover some of those correlations. It is indeed thanks to the correlation within the data that the compression is possible, because describing data with a pattern requires less information that describing uncorrelated data. For example, to represent points on a circumference, we need only the radius and the center of the circle and the curvilinear coordinates at which each point is found; in this way we have reduced the number of dimension of the representation, which is instead impossible for points disposed randomly.
In this way Autoencoders are designed to be unable to learn to copy perfectly and the model is forced to prioritize which aspects of the input should be copied, learning useful properties of the data in the process.
Usually, autoencoders minimize a loss function

$$L(x, g(f(x))) \tag{5.3}$$

which penalizes $g(f(x))$ for being different from x, such as the L^2 norm [1]of their difference. As we said earlier this pushes $g \circ f$ to learn to be an identity function if they have the capacity to do so.
A denoising autoencoder or DAE instead minimizes

$$L(x, g(f(\tilde{x}))) \tag{5.4}$$

where $\tilde{x}$ is a copy of x that has been corrupted by some form of noise.
Denoising autoencoders must therefore undo this corruption rather than simply copying their input, as we have already seen in figure 5.2.

[1]given a function φ defined over an interval X the L^2 norm is defined as $\|\varphi\|_2 = \int_X |\varphi|^2 \mathrm{d}x$

5.2 Model of the Denoising CNN

We now get back to the model of our CNN, remembering that our goal is to filter from the hitmaps all the noise hits, which are uncorrelated from one layer to the other, obtaining hitmaps with only signal hits, which, on the other hand, show correlations between the layers; we'll use the autoencoder approach for our problem, building a CNN with an encoding stage followed by a decoding stage.
During the encoding stage we alternate Convolutional layers and Max Pooling layers in such a way to reduce the dimensions of the input while increasing the depth of the image, in order to study multiple features associated with it.
At every convolutional layer we double the depth of the image and divide by two the scope on the ϕ dimension of the kernel of the convolution. With each max pooling layer we reduce by half the hitmap along the ϕ direction. We have chosen to double the depth of the image after every convolutional layer in the encoding phase in order to gradually increase the depth and obtain multiple feature maps. The size of the kernel along the ϕ dimension is halved for every consecutive convolutional layer, with the idea of progressively looking at patterns on smaller scales.
While I've tried to explain the practical reasons behind this decisions, the possible structures of the CNN are endless, and the model is still very open to changes in the future, we are just trying to understand some characteristics of the problem at hand, introducing some parameters that we can study, like for example the scope on the ϕ dimension of the kernels. In the decoding stage instead, we want to come back to the original shape of the input, so we will alternate convolutional layers with UpSampling layers; at each convolutional layer we halve the depth of the image, using the same kernel for each layer, and with each upsampling we double the ϕ dimension of the image.

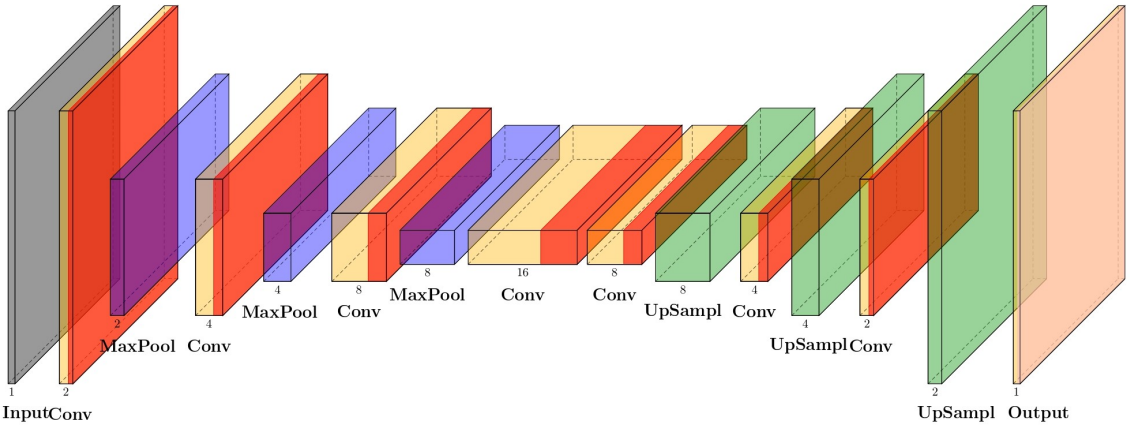


Figure 5.4: Graphical representation of the network.

Table 5.1 shows the structure of one of the CNNs that we'll be using for denoising.

Layer type	**Output shape**	**Kernel shape**	**Parameters**
Input	(800, 8, 1)	-	0
Conv2D	(800, 8, 2)	(64, 6)	770
	Activation: ReLU, padding: *same*		
MaxPooling2D	(400, 8, 2)	-	0
Conv2D	(400, 8, 4)	(32, 6)	1540
	Activation: ReLU, padding: *same*		
MaxPooling2D	(200, 8, 4)	-	0
Conv2D	(200, 8, 8)	(16, 6)	3080
	Activation: ReLU, padding: *same*		
MaxPooling2D	(100, 8, 8)	-	0
Conv2D	(100, 8, 16)	(8, 6)	6160
	Activation: ReLU, padding: *same*		
Conv2D	(100, 8, 8)	(4, 2)	1032
	Activation: ReLU, padding: *same*		
UpSampling2D	(200, 8, 8)	-	0
Conv2D	(200, 8, 4)	(4, 2)	260
	Activation: ReLU, padding: *same*		
UpSampling2D	(400, 8, 4)	-	0
Conv2D	(400, 8, 2)	(4, 2)	66
	Activation: ReLU, padding: *same*		
UpSampling2D	(800, 8, 2)	-	0
Conv2D	(800, 8, 1)	(3, 1)	7
	Activation: ReLU, padding: *same*		
Model total			**12915**

Table 5.1: CNN model with $\text{kernel}_{\phi,\,\text{start}} = 64$ and $\text{kernel}_{\phi,\,\text{end}} = 8$.

We start by training this neural network on a sample of events with noise hits and a single signal track, in order to study the performance of the network without any dependency on the number of tracks present in the training sample. Later we'll also train and apply the algorithm on multi-track events, since in the application of the algorithm on jets we'll for sure be dealing with multiple tracks. For our studies we'll be using as optimizers both RMSprop and Adam, and we'll compare the results obtained in the two ways.

When generating the samples to use for the training of our neural network we'll be generating the tracks with a flat distribution in p_T in order to allow the net to scan through the p_T interval in an homogeneous way without introducing biases in the learning process. While the task of the network and its structure are clear, I still have to talk about which Loss will be used for the training; first of all I need to clarify some aspects about the input and the target hitmaps.
We have seen in chapter 3.1 that every hit added to the hitmaps has a label with distinguishes signal from noise hits, we want this difference to be seen also by the network in order for it to learn to distinguish between signal and noise.
So the bins of the target hitmaps will have:

- **Value 2** if the bin contains a signal hit.
- **Value 1** if the bin contains a noise hit.
- **Value 0** if the bin contains no hits.

At the input level we have no knowledge of the nature of the hits, and so the bins of the input hitmaps will have instead:

- **Value 1** if the bin contains either a signal or a noise hit.
- **Value 0** otherwise.

Ideally we would like the output of the network to have bins with:

- **Value 1** if the bin contains a signal hit.
- **Value 0** otherwise.

We calculate the Loss in the following way:

1. We take the target hitmap and subtract 1 from each pixel, in this way signal hits will have value 1, noise hits will have value 0 and bins where no hit is present will have value -1.
2. We subtract these hitmaps with the output hitmaps obtained from the network.
3. From the resulting hitmap we consider only the values of the bins that were either signal or noise in the target hitmap and calculate the Mean Square of these values.

In a summarizing formula the Loss is:

$$L(y, \hat{y}) = \frac{1}{N} \sum_{\text{(i, j) s.t. } y(i,j)>0}^{N} (\hat{y}(i,j) - (y(i,j) - 1))^2 \tag{5.5}$$

where $\hat{y}$ is the output of the network, y is the target and N is the number of bins occupied by either signal or noise hits in the input.

5.3 Analysis of the Output of the CNN

Once the training is complete, the neural network is ready to be used for inference on some testing samples. As we previously discussed, the output of the CNN will be an hitmap where each pixel will possess a label value, for the algorithm to work properly, we would like for the the bins that represent signal hits to have a value close to 1, while the bins corresponding to background hits should be removed from the picture and possess a label value close to zero.
The Loss of the network is computed on the bins where either a signal hit or a background hit was present in the input image, but the output will show values different from zero also for the bins that had no signal. The loss must not depend on the output value of those bins, as we already know that they contain neither signal nor noise. Anyway the model will output whatever values are obtained by the network calculation, but these values have no meaning for us.
So we need to apply a mask to the output, setting to zero the value for the bins where no hit was present in the input image. In order to show what I'm talking about, I take the CNN showed in table 5.1, and train it with the Adam optimizer until value of the loss remains unchanged for 3 epochs. We use a training dataset of 1 million events with a single curved track of p_T values between 0.5 and 50 GeV, with the angular densities per layer shown in figure 3.8. We then apply the algorithm on a test sample of 20k single track events with the same characteristics. All the results shown in the remaining of this section are computed from the output of the network that we defined, acting on this test sample.
In figure 5.5 I show the output of the inference of the network on a single curved track event, before and after applying the mask.

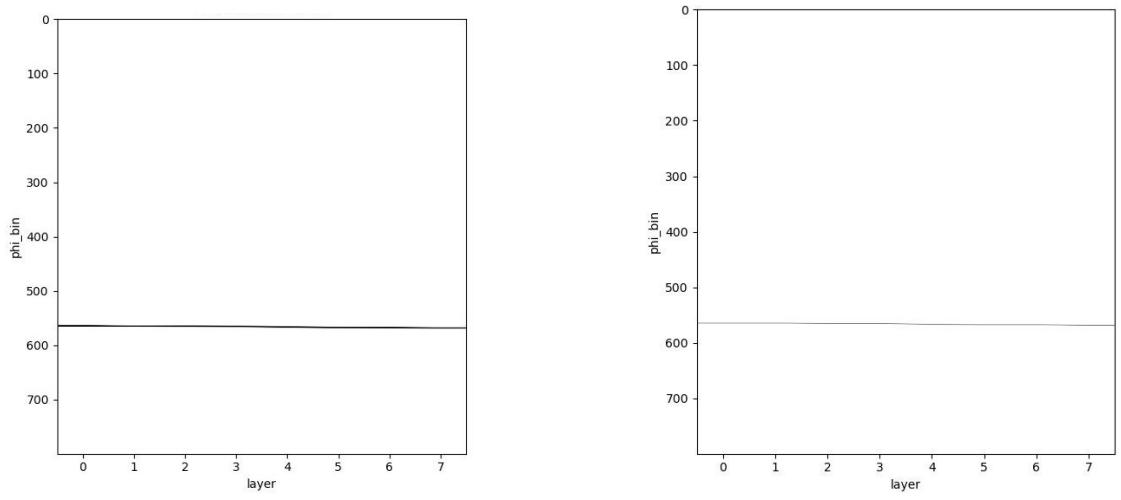
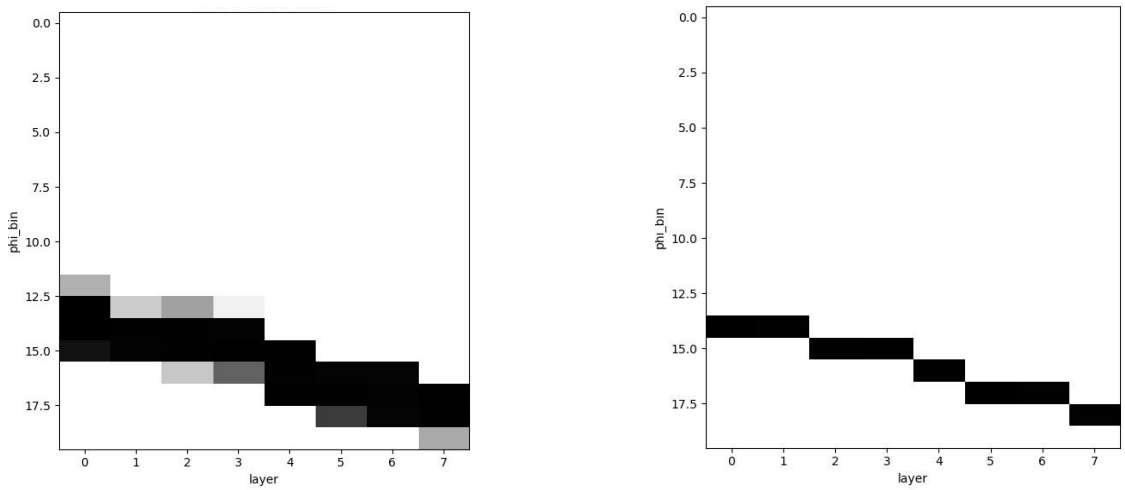


Figure 5.5: Display of the output of the CNN without the mask (left) and of the output with the mask applied (right).

We now zoom on the ϕ bins near to the tracks so that we can see better:



We see that also some bins near the track have a value close to 1, so it's important to apply the mask to the outputs, because we know that these bins didn't register any hit for this event. As a first figure of merit to judge whether the model is correctly performing its task or not, we now look closer to the weights of the output bins and examine their distribution, both for signal and background hits.
In order to do so we take the inference on 20k events of single track, using the model previously defined, and we plot the distribution for the inferred value on the bins, separately for hits we know to come from signal and from background, building the following histogram:

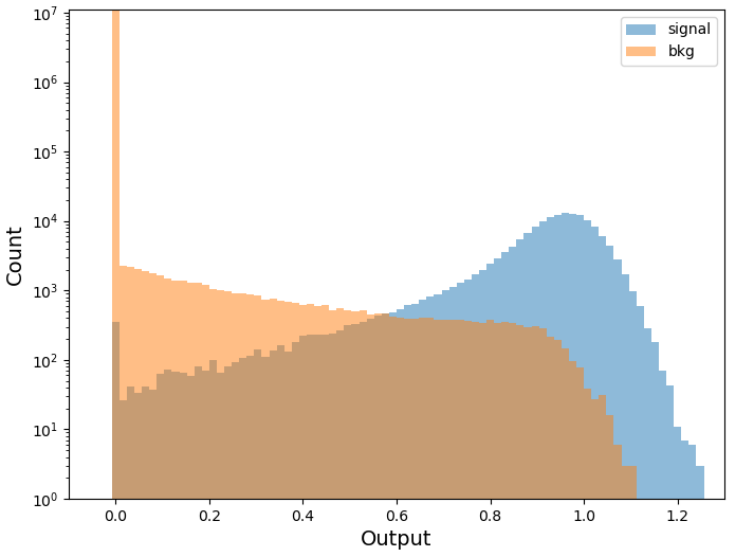


Figure 5.6: Distribution of the output values of the CNN.

We can confirm that the network is working well: most of the signal hits take an output weight close to 1 and most of the background hits have weight equal to zero.
We see that some weights are bigger than 1, that's because the nodes of the last layer of the network utilize the ReLU activation function.
A way to constrain the output to be within $[0, 1]$, could be using a sigmoid as the activation function of the last layer; I have carried out multiple trainings utilizing the sigmoid but I've noticed that most of the times the training didn't reach convergence, ending up with an algorithm that hadn't learned the task.
This is probably due to the Vanishing Gradient problem: indeed when the input of the sigmoid is either close to 1 or 0 the derivative of the activation function becomes very close to zero, making it more difficult for the weights to be updated properly, hence slowing the learning process. So, in the end, we utilize a ReLU for the last layer of our algorithm, at the price of having a weight distribution going beyond 1.
In order to discriminate between the two distribution shown in plot 5.6, we apply a Cut choosing a value between 0 and the maximum value of the weights, which is around 1.2, so that bins with an output weight above the cut are classified as signal, while the bins below are discarded and considered as background.
Using a cut value equal to zero, we expect to reconstruct all the hits present in the input image, while if the cut value gets close to 1, we expect the reconstructed hits to be signal hits, but most likely not all the signal hits will survive the cut.
It is then important to study the performance of the network changing the value of the cut, in order to understand which value is the most suited for our goals.
So we define some interesting figures of merit :

$$\text{True Positive Rate (TPR)} = \frac{\text{number of signal hits passing the cut}}{\text{total number of signal hits}} \tag{5.6}$$

$$\text{False Positive Rate (FPR)} = \frac{\text{number of background hits that make the cut}}{\text{total number of background hits}} \tag{5.7}$$

$$\text{Purity} = \frac{\text{number of signal hits passing the cut}}{\text{total number of hits passing the cut}} \tag{5.8}$$

The TPR is telling us which percentage of the signal hits passed the cut, the FPR refers to the percentage of background hits that are mistakenly classified as signal and finally the purity gives us a measure of the contamination of False Positive hits on the actual signal hits. The TPR coincides with the efficiency ϵ of the algorithm and the FPR is also called background efficiency ϵ_{bkg}.
The ideal algorithm would have both efficiency and purity equal to 1, with all the signal hits passing the cut and all the background hits rejected. In reality we'll need to settle for a compromise between the two values, remembering that at the Fast Tracking stage we need to prioritize efficiency since the following stage of Precision Tracking will deal with further corrections.
In order to obtain a more global view of the performance of the algorithm, we introduce the ROC curve, standing for Receiver Operating Characteristic.

To plot the ROC curve we scan the values of the cut from 0 to 1, and for each one we compute the TPR and the FPR, and then plot the points obtained, where the FPR values are on the x-axis and the TPR values are on the y-axis.
From this graph we can compute the AUC, or Area Under Curve, the closer this index is to 1 the better is the performance of the classifier. We obtain the plot in figure 5.7.

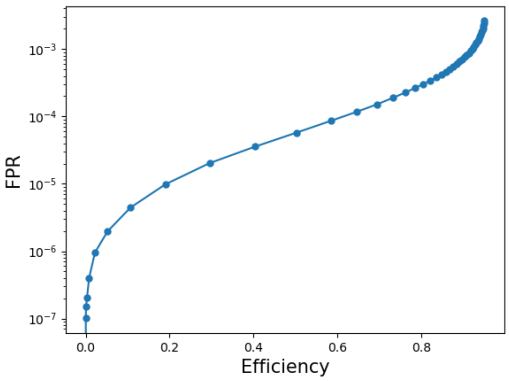


Figure 5.7: ROC with ϵ_{bkg}.

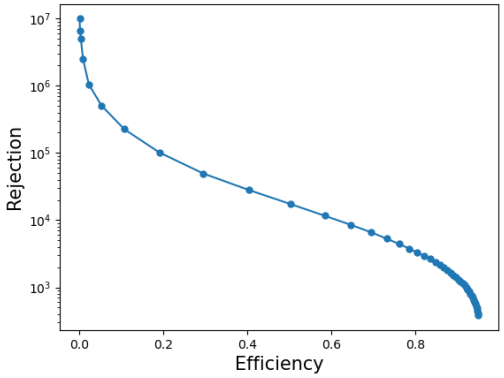


Figure 5.8: ROC with the rejection $\frac{1}{\epsilon_{bkg}}$.

An equivalent definition of the ROC curve has the TPR values on the x-axis and the Rejection Rate values on the y-axis: the rejection rate is defined as the reciprocal of the efficiency for the background, and tells us every how many background hits one is mistakenly identified as signal. With this new definition, we obtain the graph in figure 5.8.

When scanning the cut values we can also compute, for each cut, the average of the efficiency and the purity for the events in the test sample. We obtain the plots in figure 5.9. We observe that the efficiency reaches the highest value of about 95% when the cut value is zero; we would have expected an efficiency of 100%, but we obtain a lower value because some of the bins containing signal hits have an output weight of exactly zero, and consequently aren't considered in the analysis, since the cut considers values strictly greater than zero. The efficiency is a decreasing function of the cut value, going to zero when the cut equals 1. The purity instead starts from a low value and then increases, reaching the maximum for a cut of around 0.8, and then decreases going again to zero; this is due to the fact that with a high cut value we'll have events where no hit survives the cut, and in this situation we expect the purity to be zero.

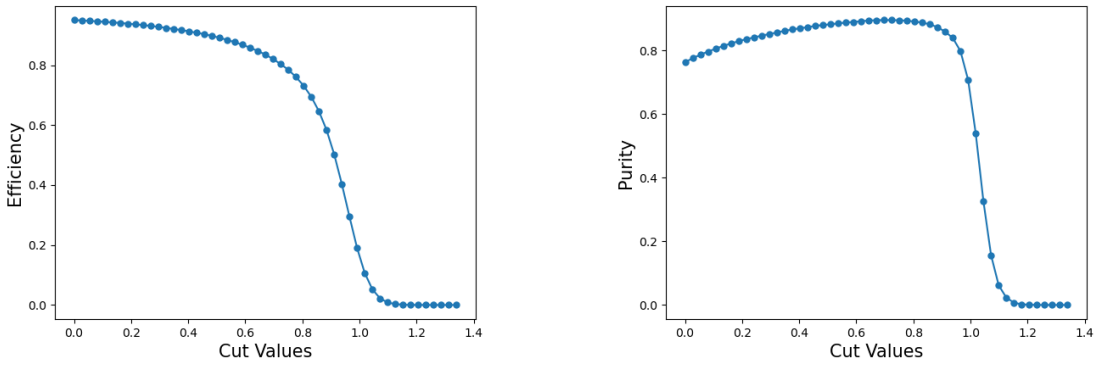


Figure 5.9: On the left the efficiency and on the right the purity, as functions of the cut values.

We now define the concept of "Working Point" (WP), which corresponds to a value of the cut such that the efficiency of the algorithm equals a specific value; for example in this case using a WP of 90% means to utilize a cut value around 0.45 so that the average efficiency on the events in the testing sample is about 0.9. On the left of figure 5.10 I sketched how the working points relative to 80%, 85% and 90% are found.

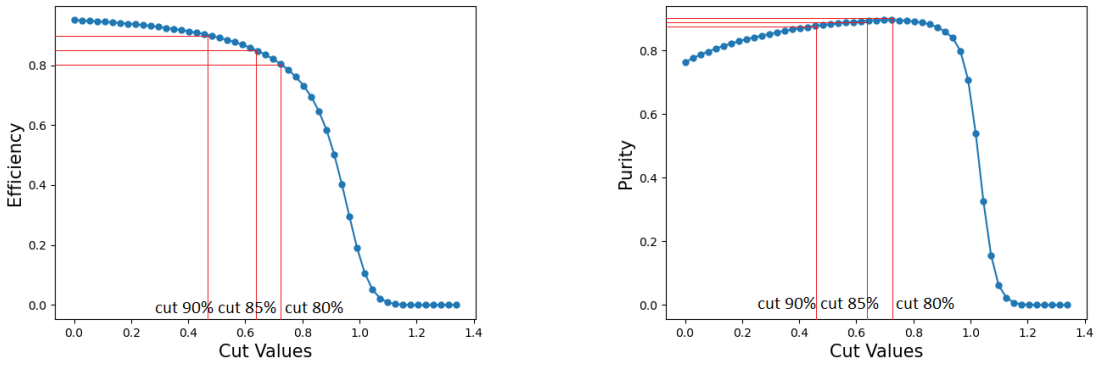


Figure 5.10: Sketch of how we interpolate to find the WPs.

In the plot on the right of figure 5.9 we see how we compute the purity values corresponding to the cut values obtained choosing the WPs, and in this instance there is no considerable gain in the purity of the output when choosing lower efficiencies.
Each point that appears in the plots of figures 5.9 and 5.10 is an average value over the entire testing dataset, but of course we expect that these values won't be homogeneous for all signal hits, and that multiple additional factors will impact he performance.

In the following we check if the figures of merit that we have just introduced have a dependency on these quantities:

- the p_T value of the track,
- the layer on which the signal hit is detected,
- the ϕ angle of the signal hit detected.

To answer the question we examine these three cases separately.

5.3.1 Performance vs p_T

We are wondering how the p_T value of the track influences the performance of the algorithm, and we would like to plot the trends of the efficiency and the purity, obtained for the output of the algorithm, as functions of p_T.
First of all, we expect the algorithm to have learned to recognize tracks with high transverse momentum, since a straight line on the hitmap is a pattern that could be easily spotted, even with the high values of noise density that we're dealing with.
On the other hand, tracks with lower p_T will be more spread on the bins in the ϕ direction and will show a discontinuity between the innermost and outermost layers, making it a harder task to distinguish them from the background.
We now plot in Fig. 5.11 the efficiency and the purity in the p_T range of $[0.5\text{ GeV}, 50\text{ GeV}]$, for the three WPs that we have just defined: 80%, 85%, 90%. Please note that, here and in the following, efficiency and purity plots will not show error bars, as our main goal is to compare different trends on the same sample, so statistical fluctuations do not spoil the result of our investigations.
Our intuition was right, the efficiency of the algorithm gets worse for lower values of p_T. Furthermore, when choosing a higher working point, the plateau value of the purity decreases, while instead, for lower values of p_T, the higher WP also has better purity.
In the plot of the efficiencies we notice that the three functions reach, for high momenta, a plateau slightly above the value of the WP. That's because the target efficiency used to define the WP is the average over the entire range, so if the efficiency is below target for low-p_T tracks, it has to be above target for high-p_T ones.
When confronting different models there is an issue if we use these plots with the same WP: the worse model will have lower efficiencies for low momenta, but since the overall average has to equal the WP, the function will plateau to higher efficiencies at high p_T, in order to balance the average.
We would like to confront models that plateau at the same values, so that we equalize the efficiency curves for high p_T and the performance for low momenta, gives us information about the pattern recognition power of the model.
To do so we compute the WPs only on the subset of events with $p_T \in [40\text{ GeV}, 50\text{ GeV}]$, where the tracks are basically straight, and then we utilize the cut values obtained to compute the performances on the full p_T interval of $[0.5\text{ GeV}, 50\text{ GeV}]$.

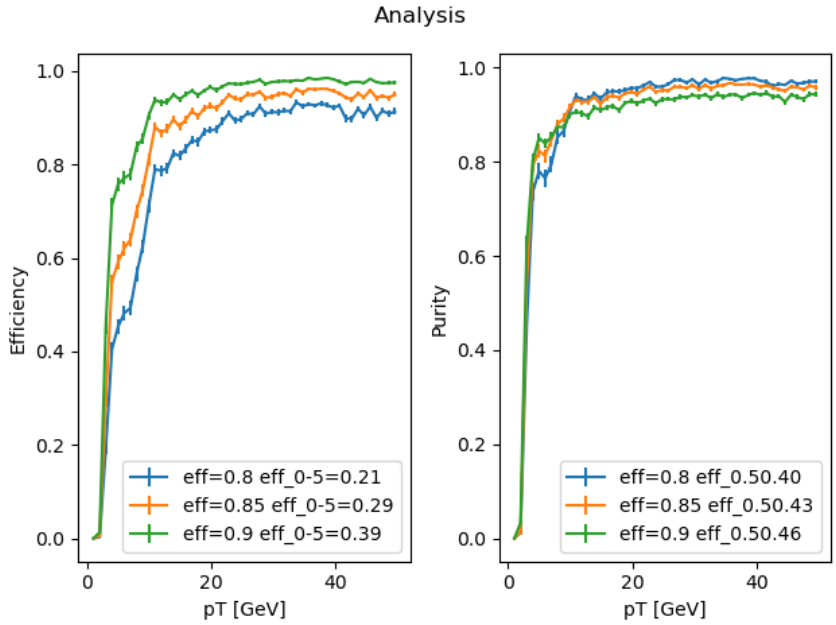


Figure 5.11: Efficiency and purity as functions of p_T, WPs computed on the entire sample.

So we take a testing sample of events produced with the same angular densities of noise hits, but with tracks of momenta within $[40\ \text{GeV}, 50\ \text{GeV}]$, and compute the efficiency and purity for the different values of cut, plotting the results in figure 5.12.

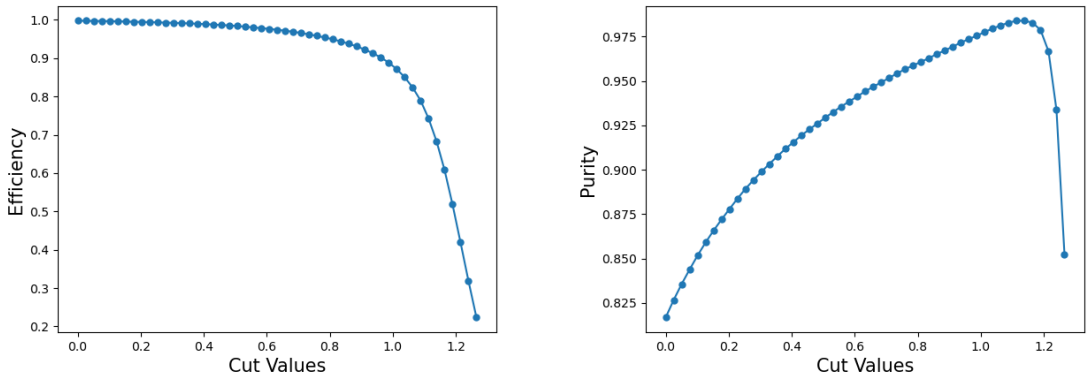


Figure 5.12: Efficiency and purity as functions of the cut values, for the $p_T \in [40\ \text{GeV}, 50\ \text{GeV}]$ sample.

As we expected the performance on this sample is better than on the full-range samples, since high-p_T tracks are easier to identify; we thus use these ROCs to define some WPs,

namely 85%, 90% and 95%. For these WPs we plot the dependence of the performance on p_T in figure 5.13.

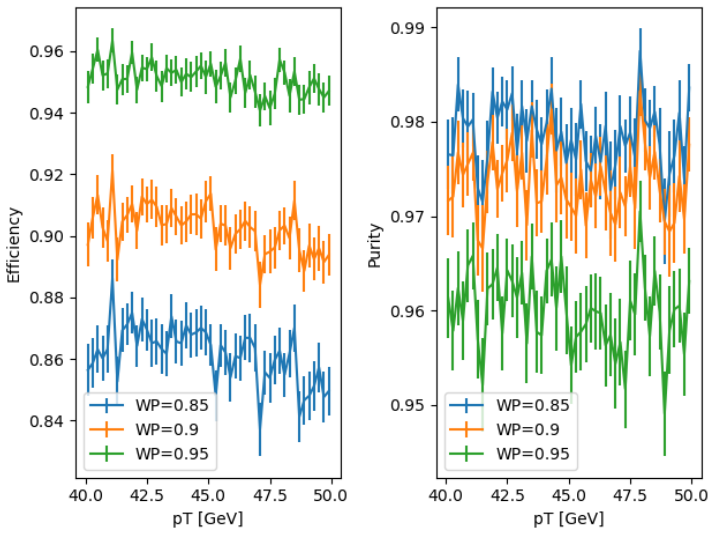


Figure 5.13: Efficiency and purity as functions of $p_T \in [40\text{ GeV}, 50\text{ GeV}]$, WPs computed on the sample itself.

We observe that in the range $[40\text{ GeV}, 50\text{ GeV}]$ the performance stays more or less constant; this is coherent with the fact that the tracks above $p_{T,\text{ threshold}} = 39$ GeV aren't distinguishable for the network, as we commented in section 3.1.
We take note of the cuts utilized for the three WPs and use the same ones on the $[0.5\text{ GeV}, 50\text{ GeV}]$ sample. Plotting the dependence on p_T , we obtain figure 5.14.
As we expected the efficiency value at the plateau of the functions matches the values of the WPs. Now that we have found a more appropriate way to compare the p_T performance of different models, we also look for a measure so that the comparison can be more quantitative.
A possibility is to compute the Area Under Curve (AUC) for the plots like the one shown in figure 5.17, so that, if the area is calculated for the same p_T range, the model with the higher AUC is the one that performs better. With his approach though the AUC is dependent on the range of momenta for which we are computing the Area, a more flexible solution is to get the average value of the efficiency and the purity for the bins in the desired range, so that this measure is confined between 0 and 1. For instance we focus on the interval $p_T \in [0.5\text{ GeV}, 5\text{ GeV}]$, since we largely care about low momenta performance.
Another possible measure is the value of the turn-on for p_T, which is the value of the momentum at which the efficiency reaches a threshold that we decide, for example the 80% of the efficiency at the plateau. For the model that we have looked at thus far, focusing on

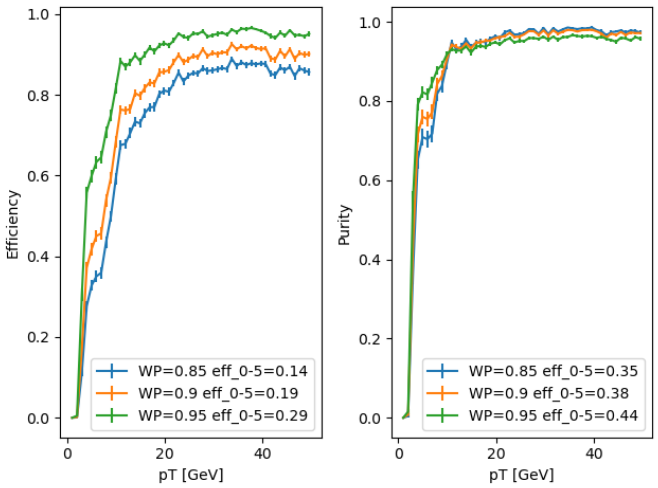


Figure 5.14: Efficiency and purity as functions of $p_T \in [0.5 \text{ GeV}, 50 \text{ GeV}]$, WPs computed on the $p_T \in [40 \text{ GeV}, 50 \text{ GeV}]$ itself.

the 98% working point, we obtain:

$$p_{T,\text{ turn-on}} = 9.9 \text{ GeV} \tag{5.9}$$

Since the p_T curves shown on the left on figure 5.14 are approximately monotone, this measure is well defined, and we can say that models with a lower value of $p_{T,\text{ turn-on}}$ perform better.

5.3.2 Performance vs Layer

We want to examine if the performance changes for different layers of the detector simulated with DiTTo. In section 3.1 we talked about the discontinuity introduced in the hitmaps between the 4th and 5th layer, because of the distance between the layers that play the role of the pixels and those that simulate the SCT; we are curious to see if this translates in a drop of performance between the two layers, or if the network is able to recognize the pattern despite the jump in the hitmap.
Similarly to what we did for the analysis for p_T, we compute efficiency and purity restricting ourselves to a single layer at a time. Keeping the WPs computed for events in the range $p_T \in [40 \text{ GeV}, 50 \text{ GeV}]$, we obtain plots like the one shown in figure 5.15.
First of all we notice that, for the efficiency, the values shown are lower than the corresponding WPs; that's because the WPs are obtained from high momentum tracks while, in this plots, the hits used come from tracks within the p_T interval $[0.5 \text{ GeV}, 50 \text{ GeV}]$, so that the efficiencies are an average over the full range of momenta.
We also notice that for the first four layer the efficiency gradually decreases, and we see a similar behaviour also for the last four layers, with the last one being the worst performing

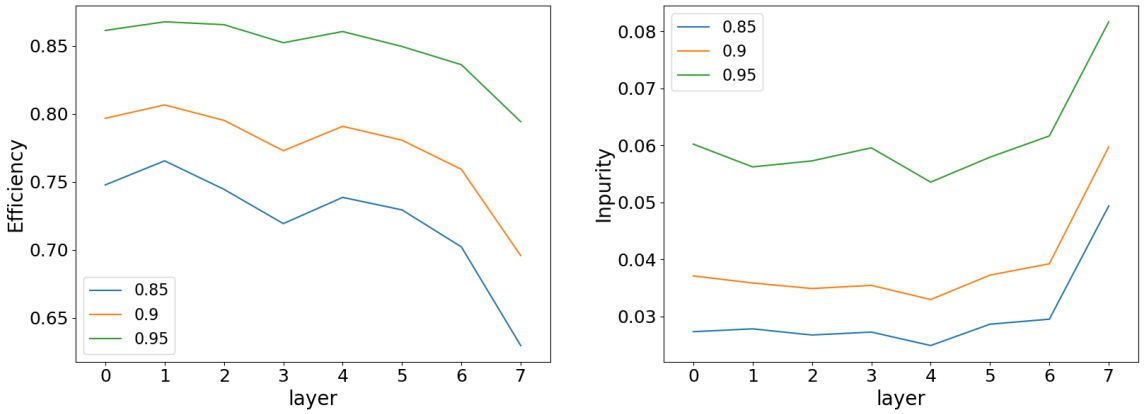


Figure 5.15: Plot of the average efficiency (left) and the average inpurity (right) computed for hits in each layer.

layer, so that for the 95% WP there's a reduction of around 10% with respect to the efficiency of the first layers. The values of the purity remain more or less unchanged for the first seven layers, but again for the last one we see that the inpurity of the reconstruction, defined as 1 minus the purity, increases by 2 percentage points.

5.3.3 Performance vs ϕ

We now check if the performance of the network is dependent on the ϕ value of the signal hits. First of all we need to explain in more detail how the ϕ intervals within which we generated the signal hits and the noise hits of both the training and testing sample.
As we said earlier the sample we are working with is composed of tracks with exit angle from the beam pipe $\phi_0 \in [-0.4, 0.4]$, in order to mimic the typical width of a $t\bar{t}$ jet. Since the tracks are curving, it will happen that some of the signal hits will end up outside of the said ϕ interval, and this will happen most likely for tracks with low p_T.
If we choose to ignore this fact and simply discard the signal hits with ϕ outside of $[-0.4, 0.4]$, then we'll have that only a fraction of the signal hits will be contained in the hitmap, making it considerably harder to recognize the hit due to the passing of a track from the random background. This would introduce a systematic inefficiency to the algorithm, that would affect the performance regardless of the quality of the model of the CNN.
In order to factor out this effect we generate the tracks with $\phi_0 \in [-0.4, 0.4]$, while we set the acceptance of the detector for $\phi \in [-0.8, 0.8]$, so that the noise hits will be generated in this interval and all the signal hits will be present in the hitmap.
Indeed if we consider a track with $p_T = 0.5$ GeV, we can compute the relative curvature with formula 3.3, and then using formula 3.6 , we can compute the $\Delta\phi$ value of the signal

hit on the last layer of the detector:

$$\Delta\phi = \tan^{-1}\left(\frac{1}{\sqrt{\frac{4R^2}{r^2}-1}}\right) = 0.32 \text{ rad}, \tag{5.10}$$

so we have checked that all signal hits are contained in the hitmap. We now plot the efficiency and the inpurity, as functions of the ϕ value of the signal hits, using the WPs for high momenta. We obtain the plots shown in figure 5.16.

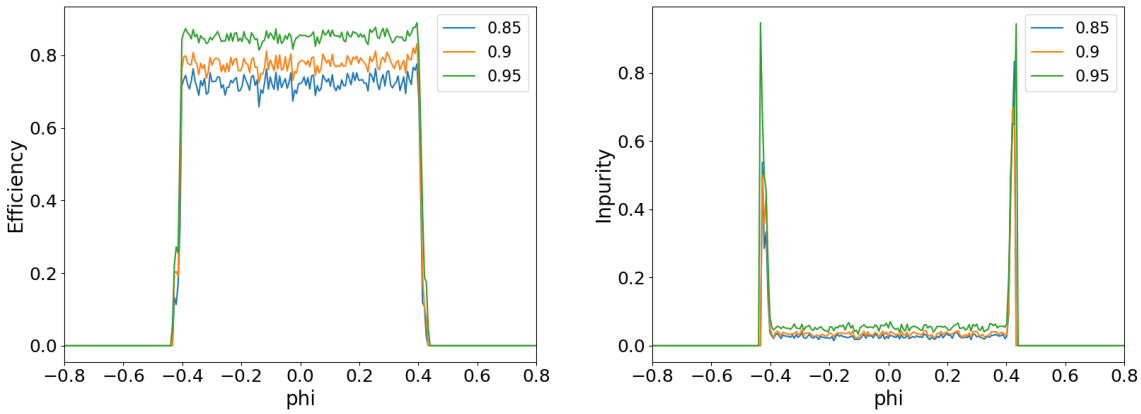


Figure 5.16: Plot of the average efficiency (left) and the average inpurity (right) computed as a function of the ϕ angle for each hit.

For $\phi \in [-0.4, 0.4]$ the values of the efficiency remain more or less constant, oscillating around the average value corresponding to the WP.
For $|\phi| > 0.4$ we notice a dramatic drop for the efficiency, which reaches zero when $|\phi| \approx 0.45$. In principle we would have expected to have values of the efficiency different from zero also at distance $\Delta\phi$ from $[-0.4, 0.4\,]$, but I think that this effect can be explained as an edge effect; during the training the intervals $[-0.4 - \Delta\phi, -0.4\,]$ and $[0.4, 0.4 + \Delta\phi\,]$ are less explored than the region $[-0.4, 0.4\,]$, because only tracks starting from the borders and with low enough p_T will end up in this regions.
I think that this may affect the ability of the algorithm to reconstruct the hits in these regions. Another effect may contribute to the drop in efficiency, because from the left plot of figure 5.15 we learned that the hits in further layers are reconstructed with a worse average efficiency, and I think that this is caused by the hits of low momentum tracks, which are not reconstructed. This effect would contribute further to reduce the efficiency in the regions outside of $[-0.4, 0.4\,]$.
Summing up, these last results comfort us on the stability of our algorithm as a function of angular track parameters, but also set a reminder for future studies, on the need to correctly deal with edge effects.

5.4 Optimization of the CNN

Now that we have introduced the figures of merit that can measure the performance of the CNN, we can study how the performance depends on the hyperparameters that we have introduced when we have defined the the model of the network and the training process, in order to choose the optimal values for the functioning of the algorithm.
Many parameters can enter in this kind of study, like for example the learning rate, the batch size, the dimensions on the hidden layers and so on, but we'll focus mainly on the scopes on the ϕ dimension of the kernels used in the convolutional layers. We've done this choice since varying this parameters we see a strong effect on the performance of the algorithm, and also because it can give us important information about the structure of the network. We'll also look at the influence of the optimizer chosen in the training, testing both RMSprop and ADAM and then comparing the results.
We'll proceed by training multiple CNNs with different architectures and looking for the one that performs the best, so that after this section we can set the model of the CNN for further analysis.
We perform this study limiting ourselves to single track events, in order to obtain results that won't be dependent on the number of tracks generated, so that we obtain information about the behavior of the algorithm in a situation well under control.

5.4.1 Kernels

We are asking ourselves how the performance of the algorithm depends on the scope along the ϕ dimension of the kernels used in the convolutional layers of the CNN. In section 5.1 we talked about the decisions we made about the structure of the CNN, in order to constrain the infinite possible models and make it possible to study the performance of the network as a function of some hyperparameters.
The hyperparameters that we explore here are the starting value and ending value of the ϕ scope of the kernels used in the encoding part of the network, that we'll call respectively $\text{kernel}_{\phi,\,\text{start}}$ and $\text{kernel}_{\phi,\,\text{end}}$. In table 5.1 we showed an example of network with $\text{kernel}_{\phi,\,\text{start}} = 64$ and $\text{kernel}_{\phi,\,\text{end}} = 8$.
We chose to start with a value for kernel_{ϕ} and halve it for each following convolutional layer, in a way to look at more macroscopic features initially and then gradually take a look to smaller scales and finer patterns. We also remember that at each convolutional step we are doubling the depth of the image and halving the size along the ϕ axis, in a way that at the end of the encoder the input has been compressed and we dispose of a high number of feature maps.
We decide two sets of values for both $\text{kernel}_{\phi,\,\text{start}}$ and $\text{kernel}_{\phi,\,\text{end}}$, and we'll study all the networks obtained by the possible combinations of these values, in order to understand which one could be the best. The two sets are:

$$\text{K}_{\text{start}} = \{16, 32, 64, 128\} \tag{5.11}$$

$$\text{K}_{\text{end}} = \{2, 4, 8\} \tag{5.12}$$

From now on with ($\text{kernel}_{\phi,\text{ start}}$, $\text{kernel}_{\phi,\text{ end}}$) we indicate a model that starts the encoder part of the network with $\text{kernel}_{\phi,\text{ start}}$ and ends it with $\text{kernel}_{\phi,\text{ end}}$.
Of course we expect the model (128, 2) to perform better than (16, 8), the first one is indeed a deeper model, with 7 convolutional layers, and consequently it will get to a high number of feature maps, hence strongly increasing the number of trainable parameters. A larger number of parameters allows the model to reach a higher capacity and expressibility, and so we expect (128, 2) to have a much better performance than (16, 8).
But there may be a threshold for the number of parameters after which there is no considerable change in increasing this number and a model may outperform a deeper one, because the architecture is more suited for the problem at hand. So we train all the combinations obtained from K_{start} and K_{end} and look at the figures of merit in order to decide the best one.
In order to fairly compare the different models we have to train and test them on the same samples, so we need to decide which sample is the most suited for the job.
Our algorithm is thought to be part of the HLT, and of course we can't change the trigger whenever the characteristics of the beam change, like for example the number of bunch interactions, so we need a robust algorithm that could have an acceptable performance for different conditions.
We try to do so, training the network on a sample with the angular densities obtained as the average over the full data-taking of a year, using the values portrayed in figure 3.9, that we report in table 5.2.

Layer	layer 0	layer 1	layer 2	layer 3	layer 4	layer 5	layer 6	layer 7
δ [rad^{-1}]	39.1	45.5	51.3	53.2	44.9	42.0	38.0	35.2

Table 5.2: Average values of the angular density of hits (δ) for an MC simulated $t\bar{t}$ sample of jets.

The training sample is the same one used in the previous section, with the angular densities of noise hits that we have just described. The networks are trained until the loss is unchanged for 3 epochs, after which we check the trend of the training error vs the validation error, to make sure that we don't end up overtraining the network.
We start by looking at the performances obtained with the Adam optimizer; later we'll also show the results relative to trainings done with RMSprop, and compare the two situations.
In figure 5.16 I present the average values for efficiency and purity computed over the range $p_T \in [0.5\text{ GeV}, 5\text{ GeV}]$ relative to the 95% working point, displaying them in a 2D grid showing all the possible combinations obtained from K_{start} and K_{end}.
Furthermore we have set the initial training parameter to $\eta = 5 \cdot 10^{-4}$ and we have trained the models until the loss is unchanged for three consequent epochs, with a maximum of 40 epochs.

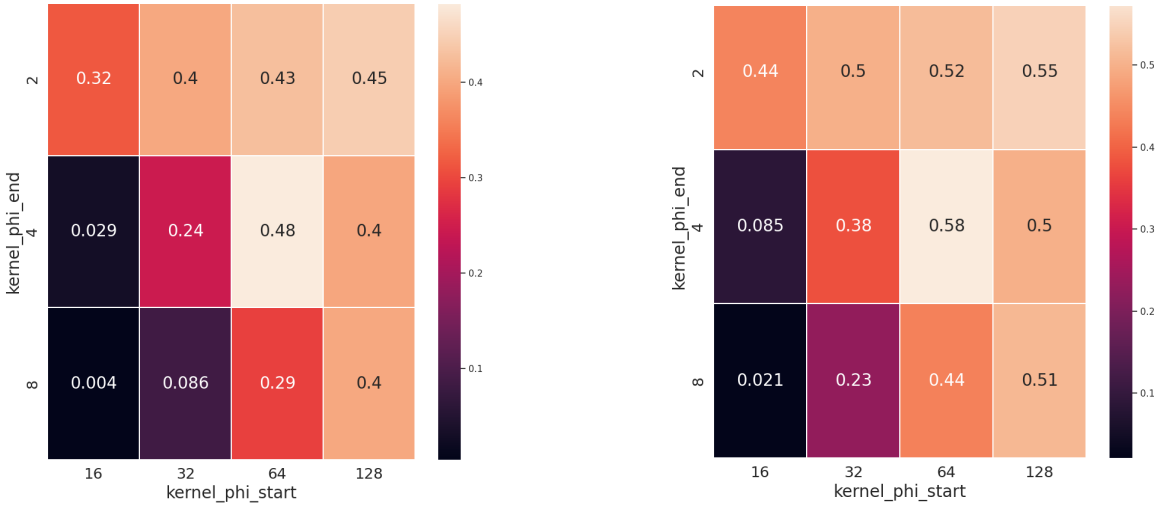


Figure 5.17: On the left the average efficiencies, on the right the average purities, both computed for the range $p_T \in [0.5\ \text{GeV}, 5\ \text{GeV}]$ and WP=0.95, using the Adam optimizer.

The general trend portrayed in the graph is that going towards larger values of $\text{kernel}_{\phi,\ \text{start}}$ and smaller values of $\text{kernel}_{\phi,\ \text{end}}$, which in both cases means having a deeper network, the performance gets better, with the exception of the (64, 4) model, which is found to be the best one among all the combinations.
Even though models like (64, 2) and (128, 4) are deeper and possess more trainable parameters, they still perform worse, and this is probably due to geometric reasons. The information that we can take out of this graph is that there's probably no gain in using a view along the ϕ direction which is wider than 64 bins, and also that going with a finer view than 4 bins may not add any information about the features of the event being analyzed.

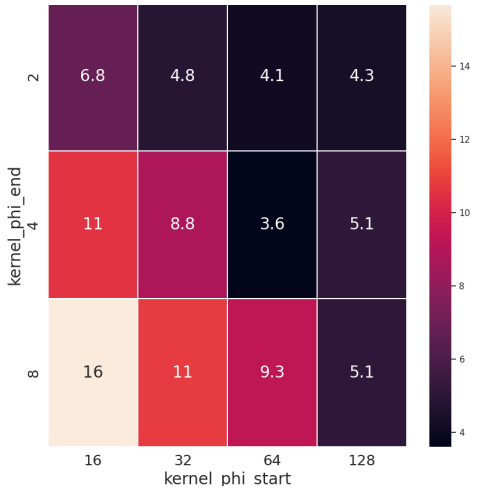


Figure 5.18: Values of $p_{T,\ \text{turn-on}}$ expressed in GeV for each model trained with Adam.

In figure 5.18 we see an analogous 2D display, where this time we are plotting the values of $p_{T,\text{ turn-on}}$ again for each model trained. Coherently to what we have seen using as measure the average efficiency and purity for low momenta, we observe that the model with the lowest turn-on for p_T is the model (64, 4), with $p_{T,\text{ turn-on}} = 4$ GeV.
Both measure indicate that the model (64, 4) is the one with the best performance.

5.4.2 Optimizer

We now propose again the same 2D displays regarding the average efficiency and purity for low momenta and the values of $p_{T,\text{ turn-on}}$, but this time we compute them using models trained with the RMSprop optimizer, so that we can compare the results with the previous displays.

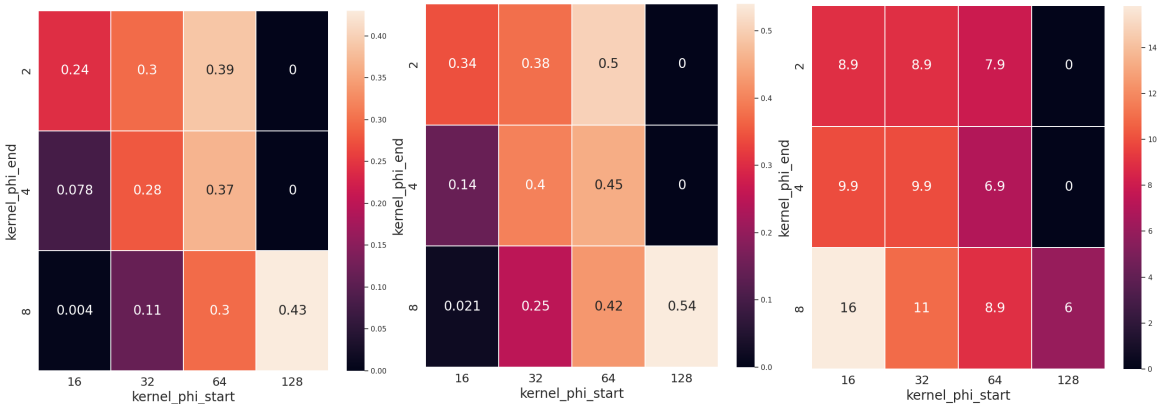


Figure 5.19: Average efficiency (left) and purity (center) computed for the range $p_T \in [0.5 \text{ GeV}, 5 \text{ GeV}]$ and WP=0.95. Also values of $p_{T,\text{ turn-on}}$ (right) expressed in GeV for each model trained with RMSprop.

According to all the metrics that we have introduced the model trained with RMSprop performs worse than the one trained with Adam.
In this case we also notice some zeros in the plot, which are not due to a bad performing model, but simply the training didn't manage to converge within the epochs of the training. This represents another weakness of the RMSprop optimizer, which we can conclude is less suited for this specific algorithm.

5.4.3 Performance vs Pile-up

Now we have done an optimization step for our model, so we can try and probe the robustness of the algorithm when we test it on samples with different characteristics from what we had in the training sample.
We'll now vary the number of primary vertices in the interaction, or pile-up, which translates in the angular densities of noise hits produced in the events. If the denoising task has been learned, we expect that modifying the density of hits, the network would still be able to filter out the noise hits, even if with a worse performance.
Indeed we want to measure how much the figures of merit that we have introduced will worsen, so that we get an idea if the network is ready to receive also more realistic inputs, where the hit density can vary to higher values.
In chapter 3.1 we looked at the average values for the angular density of hits for MC simulated $t\bar{t}$ jets, and we have plotted these angular densities for multiple values of the number of primary vertices in the events where we find the jets, in the range $[10, 40]$. We now want to test our network on different samples generated through DiTTo, with the angular densities shown in figure 3.9, in order to check the dependence of the performance of the network on the number of primary vertices in the event.
We use the model (64, 4) trained with Adam and give it as inputs the samples with hit density compatible with different values of the pile-up; in figure 5.20 we plot the average efficiency and average purity computed for $p_T \in [0.5 \text{ GeV}, 5 \text{ GeV}]$ as a function of the pile-up for each sample, and in figure 5.21 we do the same for the values of $p_{T,\text{ turn-on}}$.
We observe that the average efficiency stays more or less constant for the different values of the pile-up, with a slight decrease for the events with $\mu \geq 30$. This behavior also reflects on the plot of the $p_{T,\text{ turn-on}}$ values, which increase for high pile-up events.
The average purity instead starts around 60% and with a clear decreasing trend arrives to 30% for events with $\mu = 40$, independently from the WP chosen.

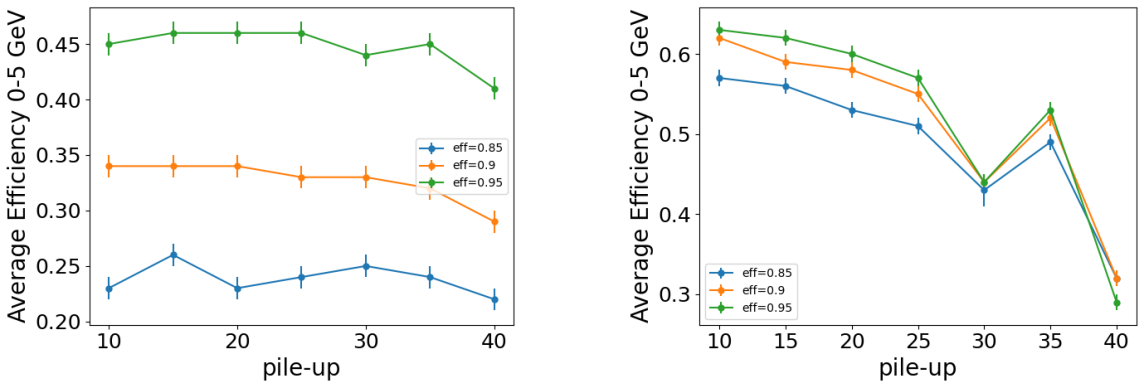


Figure 5.20: Average efficiency (left) and purity (right) in the range $p_T \in [0.5 \text{ GeV}, 5 \text{ GeV}]$ and WP=0.95, for samples compatible with different values of the pile-up.

The model shows a good robustness in reconstructing the signal hits in the output, but with higher densities of the noise more background hits are wrongly recognized as signal.
A possible way to reduce the dependency of the purity on the pile-up could be to introduce samples with higher noise hit density, so that the network could learn the task for more challenging environments.

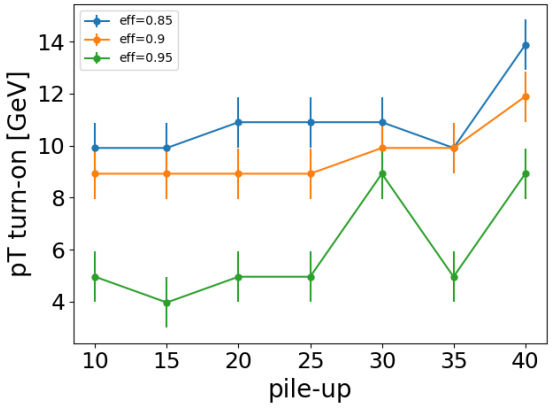


Figure 5.21: Values of $p_{T,\text{ turn-on}}$ expressed in GeV for each model trained with Adam.

We are anyway satisfied with the weak dependence of the efficiency on the noise density, since we are focusing on having an algorithm that could preserve the signal hits, and the correction of the purity will happen in further steps of the HLT.
Another important issue is the strong dependency on the pile-up for the processing time in the Fast Tracking stage, as we showed in figure 3.4; one of the guiding reasons to introduce ML-based algorithms for tracking, and in our case for filtering out noise hits, is that there should be no dependence on the hit density of the environment, since the network that performs the computation remains unchanged.
We now check this assumption for our model, plotting in figure 5.22 the processing time needed to perform the task on a single event coming from samples with different values of the pile-up. The timing is obtained with the network running on two GPUs, and is computed as the average over 20 iterations for each sample.
We notice that the timing per event, computed for both batch=1 and batch=32, has no strong dependence on the pile-up, hence giving proof to our hypothesis. We notice though way lower timings when the batch is larger, that's because GPUs can use better their capacity for parallel processing when multiple events are fed at once as the input.
In figure 5.23 we plot the average timing per event as a function of the batch size used for the inference, the testing dataset is generated with the average hit density obtained from the $t\bar{t}$ sample. We notice a decreasing trend which plateaus for batch$\approx$50 with timings in the order of 100 μs.

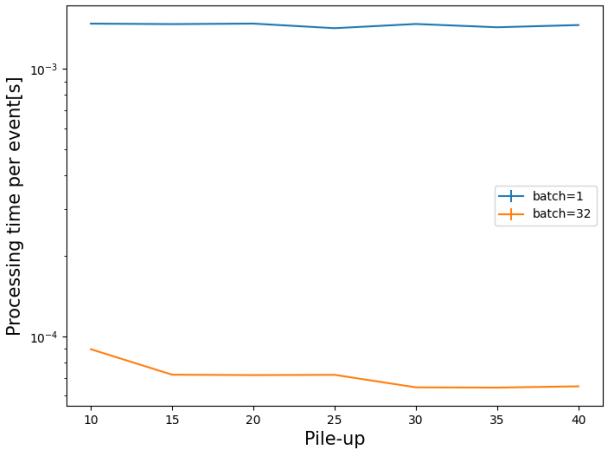


Figure 5.22: Processing time per single event for samples with different values of the pile-up.

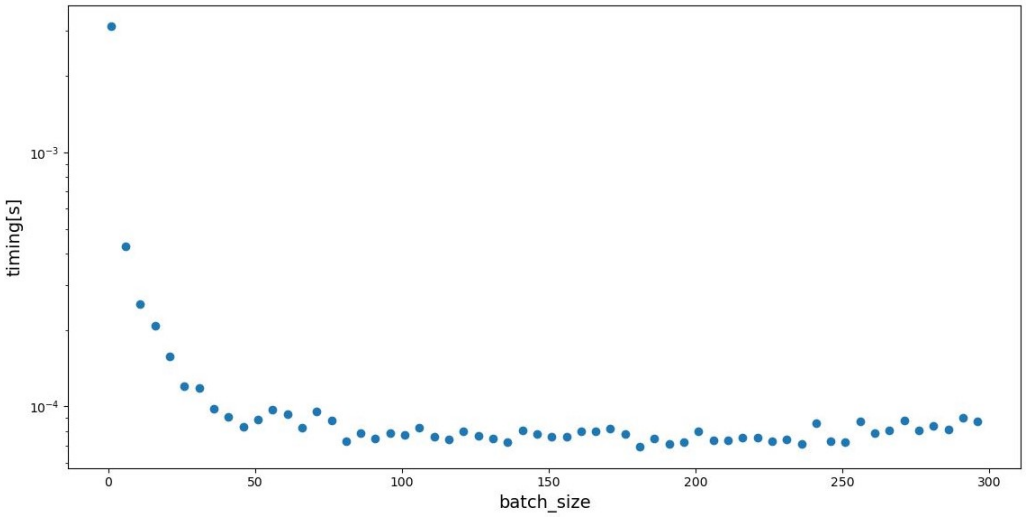


Figure 5.23: Processing time per event as a function of the batch size.

Conclusions

This thesis presents and characterizes a novel approach based on machine learning for classifying the hits in a simulated tracking detector as originating from a charged particle instead of noise. This approach is based on convolutional neural networks and has the potential to mitigate the challenging problem, and in particular the associated computational cost, of track reconstruction at the trigger level for the future high-luminosity program of the LHC.
Among the various CNN-based models described in this thesis from the possible combinations of the values for the starting and the ending kernel along the ϕ angular coordinate, I have found that the model (64, 4) is the best performing one, with an average efficiency of 48% for tracks with $p_T \in [0.5 \text{ GeV}, 5 \text{ GeV}]$, and an average purity of 58%, while granting optimal performance for tracks with transverse momentum above few GeV. To make this filtering algorithm even more suited for use at the HLT level it would be interesting to increase its efficiency for low momentum tracks, since they carry a large fraction of the useful information, especially for hadronic signatures, as we saw in figure 3.15.
On the other hand the (64, 4) model has a very simple architecture and uses less than 100k parameters, so there's definitely room for improving its performance.
I have also studied the behaviour of the model when confronted with events submerged in a high numbero of pile-up interactions; the efficiency proves very robust and stable, paying an acceptable price on the purity, which decreases down to about 30%; this is a promising result, because the observed level of background rejection in the harsher conditions would anyway save a significant amount of processing time in the subsequent fast track reconstruction step.
It must be noted that we have limited the training of the algorithm to events resembling the hit density of Run 2; training the network to even denser pile-up conditions could yield even better results.
I have also confirmed the hypothesis that the timing for a ML-based algorithm with a fixed size input does not depend on the hit density of the event, as we can see in figure 5.22. The average timing per event, found for inference on GPU is of about 100 μs, which is also a good result, to be confirmed in the future with tests on the real HLT CPU farm environment.
An extra bonus of the proposed approach is that CNN algorithms can be easily accelerated with FPGA devices, so that the timing of the denoising algorithm should be well within the specifications of the trigger system. Future prospects include anyway a careful optimization

of the time spent to convert input data coming from the detector into the matrix format handled by the CNN, and to convert back the output filtered hitmap into regions of the detector where the interesting space points appear, to seed the subsequent track reconstruction.
The approach that I have described through out all my thesis work, consists in utilizing the CNN as a filter of pile-up tracks. Even if results are not mature enough for being presented in this work, I also studies the possibility to utilize a similar approach for tasks which are even closer to the a full track reconstruction: it's infact possible to develop a CNN for the inference of the momentum of single-track events or to perform classification of the number of tracks present within the RoI.
Finally, one mandatory extension of this work is to expose the model I developed to correlated noise hits produced by simulated pile-up tracks, instead of the randomly generated background adopted so far. This study will be particularly interesting, as the correlations between background hits on different layers could on one side make them resemble more closely to the signal hits, but at the same time a clear relation among noise hits could help the CNN classifying this sort of background.

Bibliography

[1] Michael E. Peskin and Daniel V. Schroeder, *An Introduction to quantum field theory.* USA: Reading, USA: Addison-Wesley, 1995.

[2] C.P. Burgess and G.D. Moore, *The standard model: A primer.* USA: Cambridge University Press,, 200.

[3] Matteo Cacciari, Gavin P Salam and Gregory Soyez, "The anti-k_t jet clustering algorithm.," *Journal of High Energy Physics*, no. p. 063, 2008. https://doi.org/10.1088%2F1126-6708%2F2008%2F04%2F063.

[4] M. Tanabashi et al., "Review of Particle Physics.," *Phys. Rev. D*, no. 98, 2018. DOI:10.1103/PhysRevD.98.030001.

[5] The ATLAS Collaboration, "Deep Sets based Neural Networks for Impact Parameter Flavour Tagging in ATLAS.," tech. rep., CERN, Geneva, 2020. https://cds.cern.ch/record/2718948.

[6] The ATLAS Collaboration, "Deep Sets based Neural Networks for Impact Parameter Flavour Tagging in ATLAS," tech. rep., CERN, Geneva, 2020. https://cds.cern.ch/record/2718948.

[7] The ATLAS Collaboration, "Graph Neural Network Jet Flavour Tagging with the ATLAS Detector," tech. rep., CERN, Geneva, 2022. https://cds.cern.ch/record/2811135.

[8] L. Evans and P. Bryant, "LHC Machine," *ATLAS-CONF-2021-052*, vol. 3, pp. S08001–S08001, August 2021.

[9] The ATLAS Collaboration, "Measurement of the Inelastic Proton-Proton Cross Section at $\sqrt{s}$=13 TeV with the ATLAS Detector at the LHC.," *Physical Review Letters*, November 2016. http://dx.doi.org/10.1103/PhysRevLett.117.182002.

[10] Catrin Bernius, "Searching for Pairs of Higgs Bosons in the LHC Run 2 Dataset.," *Journal of Physics: Conference Series*, no. 1271, 2019. DOI: 10.1088/1742-6596/1271/1/012004.

[11] The ATLAS Collaboration, "The ATLAS Experiment at the CERN Large Hadron Collider.," *Journal of Instrumentation*, no. 98, 2008. DOI:10.1088/1748-0221/3/08/s08003.

[12] Will Buttinger, "The ATLAS Level-1 Trigger System.," *Journal of Physics: Conference Series*, no. 39, 2012. DOI:10.1088/1742-6596/396/1/012010.

[13] The ATLAS Collaboration, "The ATLAS Inner Detector Trigger performance in pp collisions at 13 TeV during LHC Run 2," tech. rep., CERN, Geneva, July 2021. https://doi.org/10.48550/arXiv.2107.02485.

[14] The ATLAS Collaboration, "Fast b-tagging at the high-level trigger of the ATLAS experiment in LHC Run 3," 2023.

[15] The ATLAS Collaboration, "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC," *Phys. Lett. B*, no. 716, 2012. https://doi.org/10.48550/arXiv.1207.7214.

[16] Peter W. Higgs, "Spontaneous Symmetry Breakdown without Massless Bosons.," *Phys. Rev.*, no. 145, 1966. https://doi.org/10.1103/PhysRev.145.1156.

[17] D. de Florian and J. Mazzitelli, "Higgs Boson Pair Production at Next-to-Next-to-Leading Order in QCD.," *Phys. Rev. Lett.*, no. 111, 2019. https://doi.org/10.48550/arXiv.1309.6594.

[18] F. A. Dreyer and A. Karlberg, "Vector-Boson Fusion Higgs Pair Production at N^{3}LO.," *Phys. Rev. D*, no. 98, 2018. https://doi.org/10.48550/arXiv.1811.07906.

[19] The ATLAS Collaboration, "Search for resonant pair production of Higgs bosons in the $b\bar{b}b\bar{b}$ final state using pp collisions at $\sqrt{s} = 13$ TeV with the ATLAS detector," tech. rep., CERN, Geneva, 2022. https://doi.org/10.1103/PhysRevD.105.092002.

[20] Elizabeth Brost and Luca Cadamuro, "Searching for Pairs of Higgs Bosons in the LHC Run 2 Dataset.," *Symmetry*, no. 14, 2022. https://doi.org/10.3390/sym14071467.

[21] The ATLAS Collaboration, "Combination of searches for non-resonant and resonant Higgs boson pair production in the $b\bar{b}\gamma\gamma$, $b\bar{b}\tau^+\tau^-$ and $b\bar{b}b\bar{b}$ decay channels using pp collisions at $\sqrt{s} = 13$ TeV with the ATLAS detector," tech. rep., CERN, Geneva, November 2021. https://cds.cern.ch/record/2786865.

[22] The ATLAS Collaboration, "HL-LHC prospects for the measurement of Higgs boson pair production in the $b\bar{b}b\bar{b}$ final state and combination with the $b\bar{b}\gamma\gamma$ and $b\bar{b}\tau^+\tau^-$ final statesat the ATLAS experiment," tech. rep., CERN, Geneva, November 2022. https://cds.cern.ch/record/2841244.

[23] The ATLAS Collaboration, "ATLAS TDAQ Phase-II Upgrade: Technical Design Report," tech. rep., CERN, Geneva, 2017. https://cds.cern.ch/record/2841244.

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

[25] John Duchi, Elad Hazan, Yoram Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization.," *Journal of Machine Learning Research 12*, no. 12, 2011. https://www.researchgate.net/publication/220320677_Adaptive_Subgradient_Methods_for_Online_Learning_and_Stochastic_Optimization.

[26] Diederik P. Kingma, Jimmy Ba, "Adam: A Method for Stochastic Optimization.," 2014. https://arxiv.org/abs/1412.6980.